(1)

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/88D-3

AN ANALYSIS OF NOISE REDUCTION USING BACK-PROPAGATION NEURAL NETWORKS

THESIS

Kevin S. Cox
Captain, USAF

AFIT/GCE/ENG/88D-3

Approved for public release; distribution unlimited.

AFIT/GCE/ENG/88D-3

AN ANALYSIS OF NOISE REDUCTION USING BACK-PROPAGATION NEURAL NETWORKS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Kevin S. Cox, B.S.

Captain, USAF

December 1988

Acknowledgements

I would like to express my deepest gratitude to my wife Cynthia, whose support and encouragement throughout my AFIT education kept my spirits high. Her reassurances were especially valuable when I began the writing phase of this research. It was during this period that I began to lose my motivation and it was Cynthia's enthusiasm that kept me going.

I would also like to thank my thesis advisor, Dr. Kabrisky, for his advice and guidance. I have never had more respect and awe for anyone than I have developed for him during these last eighteen months. His instruction has allow me to transition from a computer scientist to a "real" engineer, and my best memories of AFIT will be the times I spent with him.

Table of Contents

List Of Figures

Figure Page

Abstract

This thesis explored a new approach to filtering noise from digitized signals. A back-propagation neural network was trained to become a filter; and experiments were conducted using single sine wave inputs, multiple sine wave inputs, and human speech inputs. The networks' outputs were then compared to the original signals, and the frequency spectrum was examined to determine the network's performance.

Results indicated that the networks were indeed able to filter noise. However, the network's filtering ability was strictly limited to signals from the training set. The networks were not able to generalize enough to filter signals whose frequencies had never been encountered. The ability of a back-propagation network to filter noise from actual human speech was particularly interesting, since network performance was not significantly impacted as larger amounts of noise were used to corrupt the input signals.

The conclusion was that back-propagation neural networks can indeed be trained to become digital filters.

AN ANALYSIS OF NOISE REDUCTION USING BACK-PROPAGATION NEURAL NETWORKS

I. Introduction

Historical Perspective

According to Webster's Dictionary, noise is a disturbance that obscures or reduces the clarity or quality of a signal [Soukhanov, 1984:794]. Researchers and engineers have striven to separate signals from noise since the beginning of the communication era, and the problem continues to remain a major area of study. Currently, a typical method of eliminating noise might be to use a Weiner or Kalman filter in an effort to pass the original signal while stopping the noise component. Usually these filtering techniques distort the signal to some degree and inevitably allow a certain amount of noise through to the filter's output [Kabrisky, 1988b].

Neural networks may provide a new approach to signal filtering. Although in the past neural nets have primarily been used for pattern classification problems, recent experimental evidence suggests that neural nets may also be able to significantly reduce noise in some applications. Shin'ichi Tamura and Alex Waibel have experimented with a back-propagation neural network in eliminating noise from human speech. After analyzing the results they stated that a network "can indeed learn to perform noise reduction. Even for noisy speech signals that had not been part of the training data, the network successfully produced noise-suppressed output signals" [Tamura, 1988:553].

In addition to Tamura and Waibel's results, Bernard Widrow and Rodney Winter have pointed out an important similarity between adaptive signal processing and neural networks. Widrow and Winter point out that the fundamental building blocks of these items have the same basic architecture [Widrow, 1988:25]. This indicates that with the correct training algorithm an adaptive filter may be able to be implemented using a neural network. Fortunately, Widrow and Winter appear to have developed the training algorithm [Widrow, 1988:34-35].

Problem Statement and Research Approach

The Tamura assertion provides the basic impetus for investigating the ability of a neural network to act as a filter. The primary emphasis will be to characterize the filtering properties of a back-propagation neural network and to determine the performance of the network under varying conditions.

The testing will be done in three phases. In the first phase a neural network will be trained to filter noise from a digitized sine wave. This will allow the network to be examined while processing a very basic signal. The frequency response of the neural network will be determined, and the performance of the network will be measured as the noise to signal ratio is increased. The second phase will consist of training a network to filter noise from a signal created by summing three sine waves with different fundamental frequencies. These experiments will allow measurements of how the neural network filter performs with more complex signals and also of how the performance may degrade when filtering these signals. Finally, a neural network will be trained to filter noise from actual digitized speech to determine its effectiveness in real-world applications.

Thesis Organization

These first few pages have provided some insight into why noise filtering techniques are being researched and why a neural network will be the focus of the investigation. The next chapter provides a detailed description of back-propagation neural networks and how they work, and explains the work and results of Tamura's experiments. Chapter three presents the experiments in the order they were performed, the expected results, and what actually happened. Finally in chapter four, a analysis of the results is presented, along with a series of recommendations concerning future research.

II. Background

Description of Back-Propagation Neural Networks

Neural networks have been studied for many years in an attempt to build machines which can perform specific tasks as well as their human counterparts. Although realization of this goal is not expected for a long time, if at all, neural nets do seem to be able to "learn" how to solve certain problems [Kabrisky, 1988a]. In this research, the ability of a neural net to filter noise from digital signals is the focus of attention. In order to understand how a neural net can find the appropriate solution space to solve this problem, a detailed explanation of how neural networks behave is required.

Perceptron. The perceptron is an elementary model of biological neurons and is the most basic element of a neural network. It is comprised of three main components: *inputs, a threshold-summation unit, and an output.* Although the structure of a perceptron is simple, see Figure 1, it has the ability to divide n-dimensional space into two parts.

When an n-dimensional vector is input to a perceptron, each element of the input vector is multiplied by a corresponding element in the weight vector and the products are summed. This dot-product is computed inside the threshold-summation unit and the result is compared against a threshold value. If the result is less than the threshold value, then the output of the perceptron is -1, representing one region of the n-dimensional input space; otherwise the output is +1, representing another region of the input space. The actual equation for computing the output of a perceptron is

$$Y = f\left(\sum_{i=1}^{n} w_i x_i - \Theta\right)$$

where

$$f(\alpha) = \begin{array}{l} -1 \ \forall \ \alpha < 0 \\ +1 \ \forall \ \alpha \geq 0 \end{array}$$

[Lippmann, 1987:13]. The output of the perceptron identifies the region where the input vector resides. The regions are generated by a hyperplane that divides n-dimensional space into two regions with each region representing one class of objects [Rogers, 1988a].

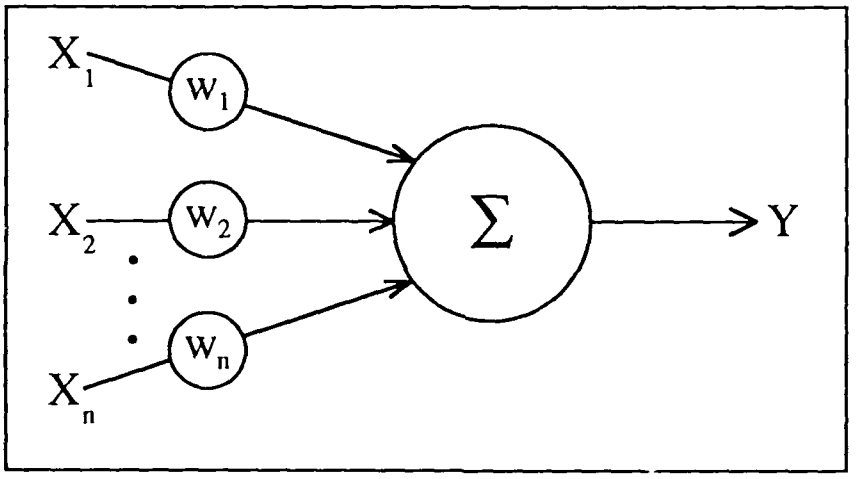

Figure 1. Diagram of a Perceptron.

Figure 2 shows how a perceptron with only two inputs may separate 2-dimensional space into regions that represent the classes A and B. The equation of the hyperplane in this example would be given by

$$x_2 = \frac{w_1}{w_2} x_1 - \frac{\Theta}{w_2}$$

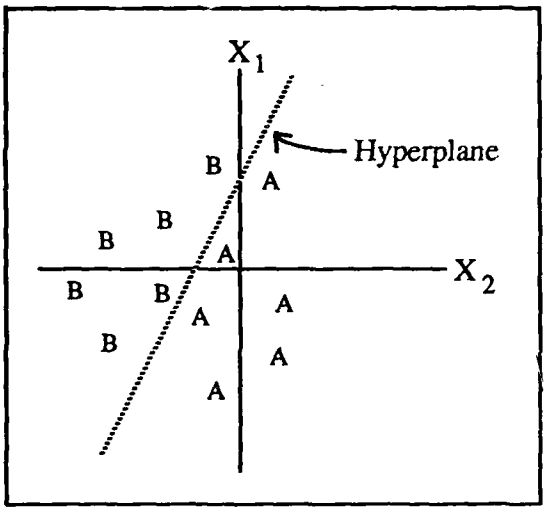

Figure 2. Two Dimensional Perceptron Decision Space [Lippmann, 1987:13].

where X_1 and X_2 make up the input vector, W_1 and W_2 are the corresponding weights, and Θ is the threshold value for the perceptron. Examination of the equation shows that the for the hyperplane to properly separate the two classes, the weights and the threshold of the perceptron must have the correct ratios. The weights can be adapted to generate these ratios by using the following procedure [Lippmann, 1987:13]:

1. Initialize the weights and the threshold to random values between -0.5 and 0.5. Ensure that the threshold value is not zero. A zero value will force the hyperplane to pass through the origin of n-dimensional space regardless of how the weights are adjusted.
2. Provide an input vector representing an item in one of the classes, and calculate the output of the perceptron.
3. If the output of the perceptron correctly identifies the inputs class, then repeat step two with a new input vector. Otherwise, compute the new weight vector using the equation

$$w_i(t+1) = w_i(t) + \eta\,[d(t) - y(t)]\,x_i$$

where W_i represents one of the weights in the weight vector; η is a learning coefficient that controls how fast the weights are adapted, as well as the stability of the perceptron; d is the desired response, -1 for one class and +1 for the other class; and x_i is the input that corresponds to the weight being updated. After updating the weights repeat steps two and three with a new input vector.

Initially, since the weights and threshold values are random numbers, it is quite unlikely that the perceptron will correctly identify the classes. However, after a number of input patterns have been presented, and the weights updated as errors occur, the perceptron will gradually move the hyperplane into a position that correctly separates the classes.

Notice that a single perceptron cannot entirely separate two classes unless they are divisible by a single hyperplane. Since many problems have a greater complexity than this a more powerful approach must be used to separate these complex classes.

Multi-Layer Perceptron. Multi-layer perceptrons are networks which contain layers of perceptrons that are not connected to the network's input or output signals. Instead, the inputs to these nodes are connected to outputs of other nodes while the output of these nodes are connected to the inputs of other nodes, see Figure 3. This type of a structure allows the hyperplanes created by individual perceptrons to be used together to build more complex decision regions.

Each node in the first hidden layer generates a hyperplane through the n-dimensional input space. By grouping these hyperplanes together, n-dimensional objects can be created by interpreting the hyperplanes to be the surface boundaries of objects. Each node in the second layer can be trained to generate an object by taking the

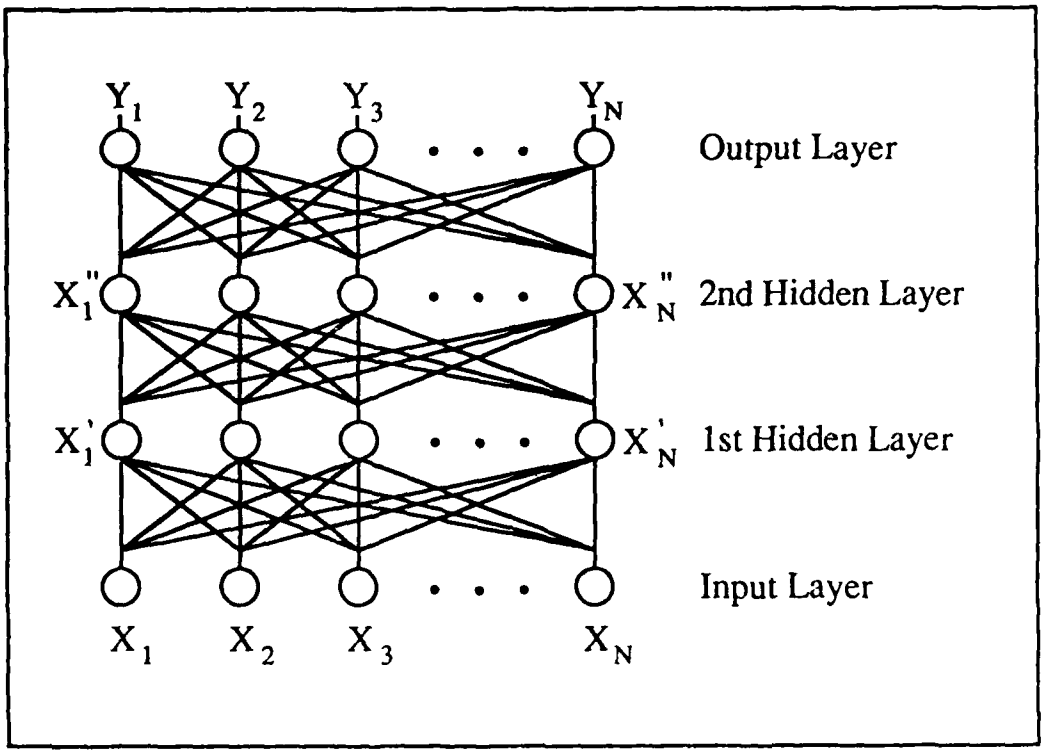

Figure 3. Multi-Layer Perceptron.

logical AND of the first layer outputs. The objects represent the decision regions generated by the second hidden layer. The output of a node in the second layer will be high only if the input vector lies within the decision region monitored by that node. The output layer then acts like the second hidden layer except it provides a logical OR capability, identifying when an input vector lies in one of several regions identified by the second layer [Lippmann, 1987:16]. This explanation does not mean that multi-layer perceptrons actually behave in this manner. In fact, very little is known about what happens inside a multi-layer perceptron and understanding this is a topic of research [Kabrisky, 1988a]. This explanation just shows that a multi-layer perceptron has enough flexibility to solve any classification problems which are separable. Unfortunately, it also

means that a multi-layer perceptron cannot outperform an optimum statistical pattern classifier if an appropriate template can be found [Kabrisky, 1988a].

An important problem with multi-layer perceptrons is training. In a single perceptron the weight vector adjustment was calculated by comparing the actual and desired outputs. Since the value of the desired outputs is not known for the hidden layers inside a multi-layer perceptron, a new training technique must be developed. To accomplish this, some changes must be made to the nodes themselves.

In the foregoing discussion, the output of a single node has been limited to zero or one. To compute the desired output for a multi-layer perceptron with hidden layers, the output function must be differentiable. This allows the training algorithm to determine how the output should be adjusted to reduce errors in the next layer of the network [Rogers, 1988b]. The differentiability can be provided by changing the output function to, say:

$$f(\alpha) = \frac{1}{1 + e^{-(\alpha - \Theta)}}$$

which is differentiable and where α is the weighted sum of the input vector for each node in the multi-layer perceptron. The weight adjustment rule that corresponds to this new output function is

$$w_{ij}(t+1) = w_{ij}(t) + \eta \, \delta_j \, x'_i$$

where W_{ij} is the weight of the connection from hidden node i to node j, η is the learning coefficient, δ_j is the error correction term, and X'_i is element i of the input vector to the node. If the node being updated lies in the output layer of the network, then

$$\delta_j = y_j (1 - y_j)(d_j - y_j)$$

where y_j is the actual output of the node, and d_j is the desired output. Otherwise, the node lies in a hidden layer of network and

$$\delta_j = x'_j (1 - x'_j) \sum_k \delta_k w_{jk}$$

where X'_j is the actual output of the node, k is the number of nodes in the layer above the node being updated, and W_{jk} is the weight of the connection between the node being updated and node k in the layer above [Lippmann, 1987:17]. This training method is called back-propagation.

Since the output of each node is a continuous function that may take on any value greater than zero but less than one, determining the meaning of the outputs is more difficult. The previous model of the perceptron had only two discrete outputs, zero and one, and each output identified one side of the perceptron's hyperplane. The meaning of outputs in a multi-layer perceptron will correspond to certainty. A value that is close to zero indicates that a node is "certain" that the input vector belongs on one side of the hyperplane, while a value close to one signals that the node "believes" that the input vector belongs on the other side. Values which move away from the zero and one extremes and towards 0.5 shows "uncertainty" in determining what an input vector signifies [Rogers, 1988a].

A network that is "cautious" or even "impetuous" can be created by controlling the decision rules for interpreting the output of the network. If input vectors are classified only when the outputs of a multi-layer perceptron lie between 0.00 and 0.05, or 0.95 and 1.00, the network is very cautious. That is the network provides classification only when it is "certain" of the input vector. However if the outputs provide classification

whenever they lie between 0.0 and 0.2, or 0.8 and 1.0, then the responses are more "impulsive" and more likely to be incorrect.

Selecting Network Parameters. The preceding explanation of multi-layer perceptrons did not provide any heuristics for selecting the proper values of the parameters. For instance, when should η be altered and by how much, how many nodes should be placed in each layer of the network, and what should the classification range be around the values zero and one? Unfortunately, the answers to these questions are not well defined. This is partially due to the way back-propagation neural networks are implemented.

For a given input vector, the outputs of a neural network are based on the weight values and the thresholds. For a given set of weights and thresholds the network will generate results with a determinable amount of error. By plotting the error against the weights a error surface can be generated. The back-propagation training algorithm forces the weight values to be altered using a first order gradient steepest descent search in a effort to move along the error surface to a local minimum [Rogers, 1988b].

η is a learning coefficient that controls how fast the network learns. Conceptually this works by limiting the amount of correction being applied to the weight vector and thus limiting the distance moved along the error surface. If a large η is selected, then larger distances will be covered along the error surface allowing the network to rapidly converge on a local minimum. However, once an area around the local minimum has been reached, a large η may cause the network to oscillate back and forth over the minimum since corrections to weight values are large. If a smaller η is selected, then a much slower traversal along the error surface will occur, allowing the network to oscillate over a region that is much closer to the local minimum [Rogers, 1988a].

Values of η that are negative will force the network to move away from local minimums. Values of η which are greater than zero and less than one will move the weight vector in the direction required to reduce the error for that input set. Finally, values of η that are greater than one will overcompensate for errors caused by a particular input. In addition, smaller values of η will do a better job of averaging past inputs and provide more stable weight estimates. Knowing this, η should be between zero and one to force the error to be reduced without over-correcting [Lippmann, 1987:13].

Selecting the number of layers in a network and the number of nodes in a layer are also choices that have few heuristics. A theorem by Kolmogorov is said to show that a three-layer perceptron could be used to create any identification function required in a classifier [Lippmann, 1987:18]. If true, this indicates that a multi-layer perceptron will never require more than three layers. However, there are two problems with this theorem. The first is that the theorem does not indicate how the weights and output functions should be selected [Lippmann, 1987:18]. The second is that the Kolmogorov theorem may have been misapplied to multi-layer perceptrons. Kolmogorov's theorem indicates that each perceptron may require a different function in order to compute another function within three layers [Ruck, 1988]. In spite of this, no one has published a problem that can be solved by a neural network with more than three layers, but not by a network with three layers or less.

The number of nodes required in each layer is equally difficult to resolve. The number of nodes in the final layer can be computed and should be the number of outputs required. The number of nodes required in the hidden layers must be sufficient to solve the problem, but not so many that the training set is memorized and the network does not generalize. One interesting result in this area has been presented by Captain Greg Tarr. He has demonstrated that more nodes are required in training a network than are actually required to solve the problem. He has trained a neural network to recognize a specific set

of input patterns. After training, he was able to remove a number of nodes from different layers without impacting the network performance. However, if he attempted to train the network with the smaller number of nodes, he was unsuccessful [Tarr, 1988].

The one area where some guidelines can be provided is in determining the thresholding values for classifying the outputs of a multi-layer perceptron. Fortunately, the classification regions do not impact the training of a multi-layer perceptron. Therefore, once the network has been trained for a particular task these values can be altered to optimize the network performance. If a network is being developed where a high degree of reliability is desired, then the threshold values must be kept close enough to the boundary values zero and one that the classifications produced by the network meet the required reliability. If a lower degree of reliability is acceptable, then the network will be able to classify a greater number of the input patterns by moving the threshold values further away from the boundary values.

Advantages of Neural Networks. Interest in neural networks has been building for several years. Although many grandiose predictions have been made and never proven concerning the ability of neural networks, there are some characteristics of neural networks that have important potential.

One characteristic is the ability of a neural network to learn and generalize. Conventional computers and algorithms cannot learn to do tasks outside of their original programming. Neural nets are not programmed for tasks, they are trained for a task by presenting them with data and responses. This allows neural networks to generalize and perform well with inputs that were not originally in the training set. In addition, generalization allows them to deal with new situations, and handle unusual inputs. Neural networks also tend to become more accurate as the amount of training increases. Without this ability to learn and generalize, neural networks lose a great deal of their

significance. It is relatively easy to write a program that saves training data and classification in a lookup table, however, to be able to extrapolate on the training data and identify input vectors that have never been seen is not a trivial task [Widrow, 1988:1112].

Neural networks are also massively parallel in their structure. This means that they can take advantage of parallel computer architectures and should be able to make decisions at very high speed. In fact, the delay between presenting an input and receiving an output is dependent upon the number of layers required in the network. For a neural net containing three layers, the delay would be three time units. If a neural network is implemented in hardware, then even greater increases in speed can be realized.

The ability of neural networks to process information at high speeds is appealing, but so is the fault tolerance of neural nets. As pointed out earlier, Captain Greg Tarr has shown that after training a neural network, a significant number of nodes can be eliminated without reducing the performance of the network. This implies that by implementing a neural network over many processors, the network should be able to sustain loss of many processors without hindering its operation.

The capabilities of neural networks to learn and generalize pattern recognition problems, their potential for speed, and their capacity to handle faults, have aroused a great deal of scientific interest. One of the research teams interested in the prospects of neural networks is Shin'ichi Tamura and Alex Waibel.

Tamura and Waibel's Experiments

Tamura and Waibel are researchers working for the ATR Interpreting Telephony Research Laboratories in Osaka, Japan. Although most neural network research to date has concentrated on pattern recognition problems; Tamura and Waibel set up a series of

experiments to test the noise reduction abilities of a net with the goal of training a back-propagation network to eliminate noise from digitized speech. They claim to have been successful in their endeavors and have published a paper explaining their work [Tamura, 1988]. This section presents a summary of that paper as it is the basis for the research done in this thesis.

Tamura and Waibel's network consisted of inputs, two-hidden layers, and an output layer, similar to the network shown in Figure 3. Each layer consisted of sixty nodes and was fully interconnected with the layer above it. Each time data was presented to the network, the output was computed, and a desired waveform output was used to generate the error signal required to update the network's weights. Since the output of the network was computed using a sigmoid function, the output was limited to values between zero and one. Therefore, the desired waveform output had to be scaled so that it was also within these boundaries. Tamura and Waibel used linear scaling techniques for this in order to preserve as much speech and noise information as possible.

The speech database used to train the network consisted of Japanese words that were spoken in isolation by professional announcers. The words were digitized at a 20 kHz rate and then down-sampled to 12 kHz. Computer room noise was recorded, digitized to 12 kHz, and added to the speech data to create noisy input waveforms with a signal to noise ratio of -20db. The original speech waveform, linearly scaled to values between zero and one, served as the desired output waveform. The network scanned each training utterance from beginning to end at a rate of 60 data points per input vector and returned to the beginning of the training data whenever the end of the data was reached. This was repeated until the squared error converged to a specified value. Figure 4 shows the result of training. The input word was "ikioi" and as can be seen from the spectrograms, the noise has been reduced while the speech spectrum has been preserved.

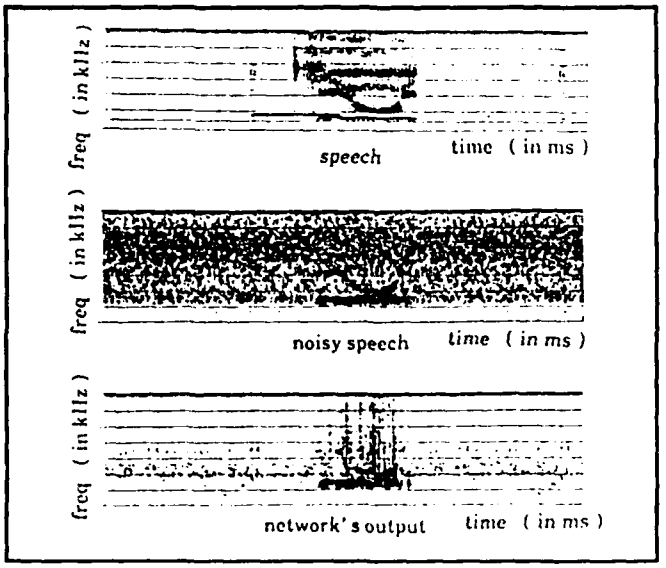

Figure 4. Spectrograms of Tamura's Training Data and Results.

In addition, if words were presented that had not been part of the training data, then the network was able to generalize and reduce the amount of noise in those waveforms. When the speech was corrupted by computer generated instead of recorded noise, the network was again able to suppress the noise. When the network's output signals were presented to listeners along with noisy speech samples that had been filtered using power spectrum subtraction, the listeners indicated that the network's output was comparable to, or better than that obtained by conventional spectrum subtraction.

In essence, Tamura and Waibel indicated that back-propagation neural networks can learn how to filter noisy signals to produce signals that have improved signal to noise ratios. These networks may also be able to generalize such that noise-suppressed signal can be produced even for data that is different from training data in both the inputs and the type of noise corrupting the inputs. This thesis will investigate this discovery in an effort to reproduce and expand upon Tamura and Waibel's results.

III. Research Experiments and Results

Chapter Overview

Certain aspects of the experiments conducted for this thesis remained the same throughout all of the research. These parameters and techniques are presented immediately to acquaint the reader with the overall methodology followed in the investigations. Following this discussion is a review of the specific experiments conducted and their corresponding results. These experiments are grouped into three categories: noise reduction in the presence of single sine waves, multiple sine wave noise reduction, and digitized speech noise reduction. The presentation of each category is started with an explanation of why the experiments were conducted, and an explanation of the general approach and assumptions that were used. Then each experiment is examined by describing the details specific to that experiment and presenting the results.

General Approach and Assumptions

Chapter two presented the basic operation of back-propagation neural networks and introduced the associated network parameters. The networks used to support this noise filtering research had several parameters which were not allowed to vary. The learning coefficient, η, was fixed at 0.3, while weights and thresholds were all initialized to random values ranging from -0.5 to 0.5. In addition, all the networks contained an input layer, two hidden layers, and an output layer; with all the layers containing an equal number of nodes. With the exception of the network architecture, all the values were based on numbers which had been successful in other neural net research at the Air Force Institute of Technology (AFIT). The network architecture, however, was based on Tamura and Waibel's research. They had used a network which contained four layers

(two hidden layers) with an equal number of nodes in each layer, and using this as a precedent the networks used in this work were constructed the same way.

To train these networks, both a target signal vector and a noise vector had to be computed. The target signal was generated differently depending upon the experiment being conducted, but the noise vector was always generated using random numbers from a Gaussian distribution with a mean value of zero. A Gaussian distribution was selected because "many naturally occurring noise phenomena are approximately Gaussian" and because "noise considerations in systems analysis are extremely difficult unless the underlying noise statistics are Gaussian" [Ziemer, 1976:202]. The actual input to the network was formed by computing the energy normalized sum of the original signal and the noise vector. The desired output of the network was the original signal linearly scaled to values between zero and one to create the target vector. The scaling was required because the sigmoid function used in the output layer of the network could only produce values between zero and one. Since the original signal's values may not have been within these bounds, the signal was linearly scaled to ensure the values were within this range. Linear scaling was selected over other scaling techniques, since that was what Tamura and Waibel had used in their experiments. The implementation of the scaling technique was dependent upon the experiment itself.

Every network was presented 500,000 training samples, and network performance was computed after every 10,000 of these samples. After training the weights and thresholds were saved to allow the network to be restored for additional testing. Two measurements were taken during each performance check to monitor how well the network was progressing: correlation and mean square error. During each performance check these measurements were computed and averaged for 100 tests.

The significance of correlation was that it provided a measurement of the similarity between the desired and actual outputs of the network. Values close to zero revealed that

the signals had no similarity, while values close to one denoted vectors which were virtually identical. Correlation was computed using

$$C = \frac{\sum_{i=1}^{n} O_i T_i}{O_l T_l}$$

where C was the correlation, O_i was the i^{th} element in the adjusted output vector generated by the network, O_l was the length of the adjusted output vector, T_i was the i^{th} element in the target signal, and T_l was the length of the target signal vector. Note that the vector O is not the actual output of the network, but an adjusted vector. For correlation to work properly, it requires that the signals being correlated be centered around zero or that a compensation term be computed. Since the networks were being trained to produce linearly scaled target signals with values between zero and one, the output of the network had to be adjusted back to the original size using the inverse of the linear scaling function.

Mean square error provided a way of determining the distance between the desired and adjusted output vectors and was a more detailed measurement of network performance. It was computed using

$$M = \frac{\sum_{i=1}^{n} (O_i T_i)^2}{n}$$

where M is the mean square error, O_i was the i^{th} element in the adjusted output vector generated by the network, and T_i was the i^{th} element in the target signal vector.

Since so many random numbers were required to support generating the initial weights and thresholds, and also to support creating noise vectors, a significance was

placed on ensuring that functions returned numbers which were truly random. It is important to note that the computers used to do the neural network simulations (VAX's with VMS operating systems) provided a system function that generated independent and random variables with a uniform distribution ranging from zero to one. This function was used as the foundation for both a uniform distribution function, with bounds α and β, and for a Gaussian distribution function with variance σ^2.

The uniform distribution function with bounds α and β was based on the formula

$$X = \alpha + (\beta - \alpha)\, U$$

where X was the random number returned by the function, α was the lower bound on the distribution, β was the upper bound on the distribution, and U was the value returned by the computer's random number generator. It should be obvious that this formula yields random values which have a uniform distribution over the desired range, since U is a independent and random variable in the range of zero to one.

Generating a random Gaussian distribution was not as intuitive. However, by using a technique explained by Averill Law, values from a standard Gaussian distribution were created using the two step process [Law, 1982:259]:

1. Generate U_1 and U_2; let $V_i = 2\, U_i - 1$ for i = 1,2; and let $W = V_1^2 + V_2^2$. When $W > 1$ repeat this step until $W \leq 1$.
2. Let $Y = [(-2 \ln W) / W]^{1/2}$; and let $X_1 = V_1\, Y$ and $X_2 = V_2\, Y$. Then X_1 and X_2 are independent, identically distributed random variables from a standard Gaussian distribution.

To convert X_1 and X_2 to values from a distribution with a variance of σ^2 the formula

$$X' = \sigma\, X$$

had to be used [Law, 1982:258]. Also, since numbers were generated in pairs using this process, noise vectors that required an even number of elements used all the random variables that were generated, while noise vectors which required an odd number of elements discarded one of the random variables generated in the last pair.

The sine waves used to train the networks were also computer generated and require some explanation. A special function was written to generate a digitized sine wave with a given frequency and phase. The number of points in the sample had to be specified. Knowing this information, the values for each point in the sample were generated using the two step process

1. $\tau = i * (360 / 1000)$
2. $S_i = (\tau * \omega) + \phi$

where τ was the time, i was the number of the sample point being computed, S_i was the value of the sample point, ω was the frequency of the sine wave, and ϕ was the phase of the sine wave. Throughout this thesis, 360° was defined to be one second. Therefore, based on the formulas just presented, the sampling rate for all the sine waves was one kHz. In addition, the frequency values provided to generate the sine waves were in Hz, while phase values were in degrees.

Single Sine Wave Noise Reduction

These first experiments attempted to remove varying amounts of noise from a single digitized sine wave. The objective was to break down the noise reduction problem into an arena where signal attributes and noise reduction abilities were easily measured. By training a network to eliminate noise from such a simple signal, the filtering properties of a neural network could be determined and the expected behavior predicted.

Approach and Assumptions. All of the networks used in this sequence of experiments had twenty-five nodes in each layer. Although Tamura and Waibel had used networks with sixty nodes per layer, that number of nodes had to be diminished to speed up the training. Networks with sixty nodes on a layer required an average of 4.5 days of VAX CPU time for training, while networks with twenty-five nodes required only 17.25 hours. Reducing the number of nodes was not expected to impact the ability of the network to filter noise, since the speech data was much more complex than single sine waves.

The linear scaling technique used to create the target output for the network was based on the amplitude of the original sine waves. Since the signals generated for the first set of experiments all had minimum values of -1 and maximum values of +1, they had to be scaled and shifted so that the values were between 0 and 1; the range of outputs that could be generated by the network. For this reason, each value in the target vector was shifted by adding 1.5, and then scaled by dividing by 3. This yielded target output values which would never exceeded the range 0.17 to 0.83, values which were easily generated by the sigmoid function. This also prevented the network from being limited to an amplitude range which was known to be valid. Instead, the network could generate an output signal with larger amplitudes than were required.

The sine waves used in these experiments also had random phase uniformly distributed over the range 0° to 360°. This was done to ensure that the networks did not "learn" to generate a sine wave with a fixed phase regardless of the input. The networks had to examine the input vector to compute the phase of the input signal. The actual frequency of the input sine waves were also distributed over a uniform range. This prevented the networks from "learning" to produce sine waves of a fixed frequency with the correct phase. Instead, the network had to compute the frequency, and the phase, in order to generate the appropriate output.

A single sine wave, regardless of frequency and phase, has a power that is directly related to its amplitude. The formula for computing the power is:

$$P = A^2 / 2$$

where P is the power, and A is the amplitude of the sine wave [Ziemer, 1976:18]. Random variables with a mean value of zero, however, have a power that is equivalent to their variance [Prescott, 1988]. By computing the power of the sine wave and the variance of the noise, the experimental signal to noise ratio could be computed. Knowledge of the this ratio was required to maintain certain levels of noise for different experiments and to compare network performance as the signal to noise ratio was altered.

Test 1 - Narrow Frequency, Fixed Amplitude, Low Noise. The first experiment consisted of training a network to eliminate Gaussian noise with a variance of 0.125, from single sine waves. The sine waves used to train the network had random frequencies, ranging from 120 Hz to 130 Hz, with a fixed amplitudes of one.

Because the sine waves used in this experiment had an amplitudes of one, the power associated with each signal was 0.5. A high signal to noise ratio was desired in the first experiment, so the noise vectors were designed to have 1/4 of the original signal's power. This implied that the noise vectors were to be generated from a Gaussian distribution with a variance of 0.125. Figure 5 presents a sample sine wave signal, the same signal corrupted by noise and used as an input to a network, and the network's output before any training had occurred. As can be seen from the graph, there is little correlation between the original sine wave signal and the signal generated by the network.

This network was then trained to filter the noise from the input signal, and the performance measurements taken during the training period were plotted. These graphs are presented in Figures 6 and 7. Initial examination of these performance graphs

indicated that the network was successful in generating the appropriate output signals, and that the network had "learned" this response after only 10,000 training samples. However, as additional performance measurements were made with the network, questions arose concerning what function the network was actually performing.

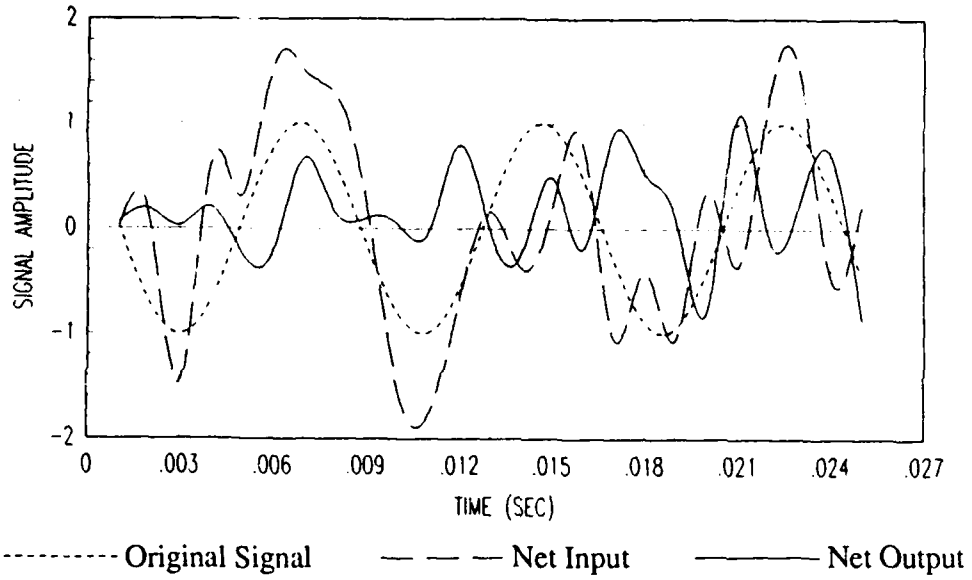

Figure 5. Test 1 - Experimental Signals Before Network Training.

The first of these measurements consisted of testing the network's filtering ability for signals ranging from 50 Hz to 200 Hz. The network was expected to work very well over the range 120 Hz to 130 Hz since those were the frequencies used in the training set. For signals outside this range, network performance was expected to drop off as the signal's frequency moved further from the training frequencies. Some of the test signals used and their corresponding results are presented in Figures 8, 9, 10, 11, and 12.

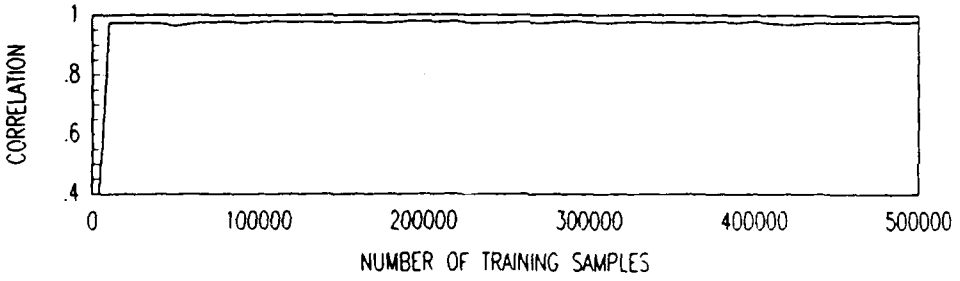

Figure 6. Test 1 - Signal Correlation During Training.

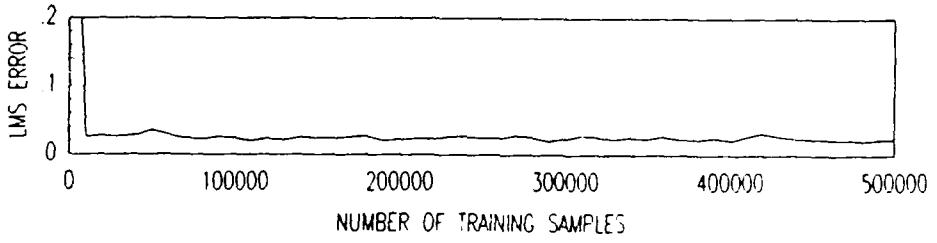

Figure 7. Test 1 - Signal Mean Square Error During Training.

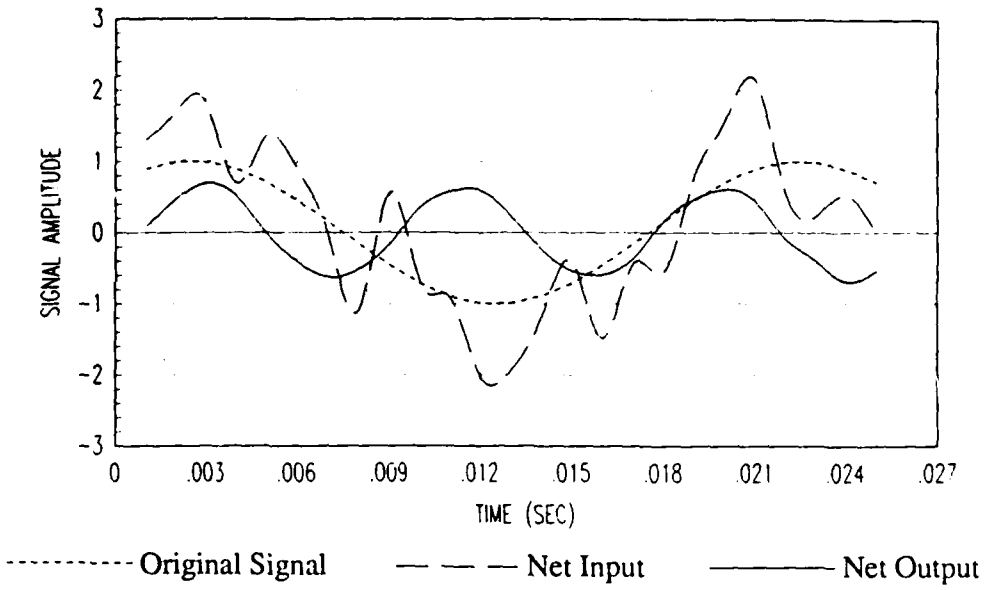

Figure 8. Test 1 - Network Behavior with 50 Hz Experimental Input Signal.

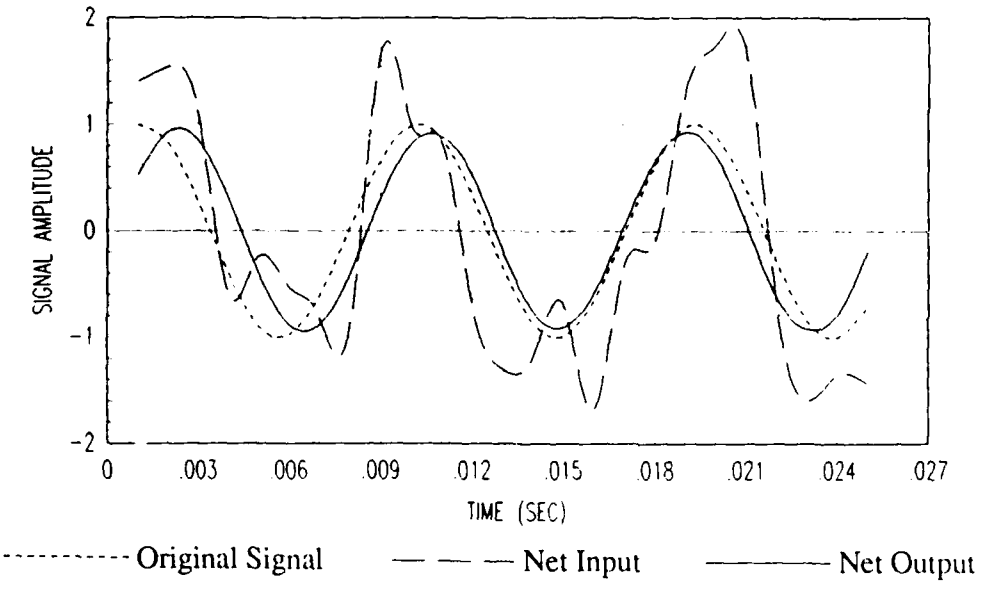

Figure 9. Test 1 - Network Behavior with 110 Hz Experimental Input Signal.

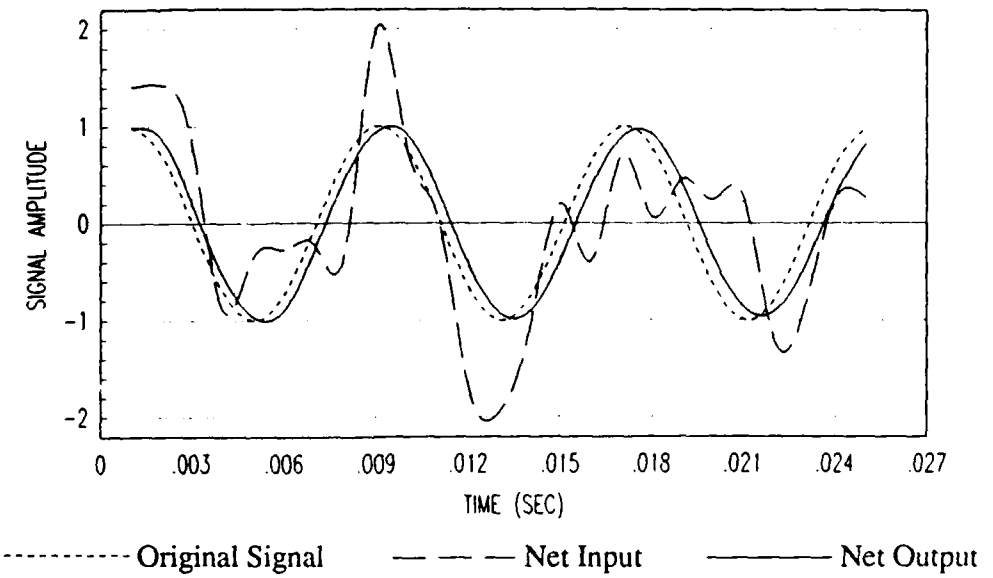

Figure 10. Test 1 - Network Behavior with 124 Hz Experimental Input Signal.

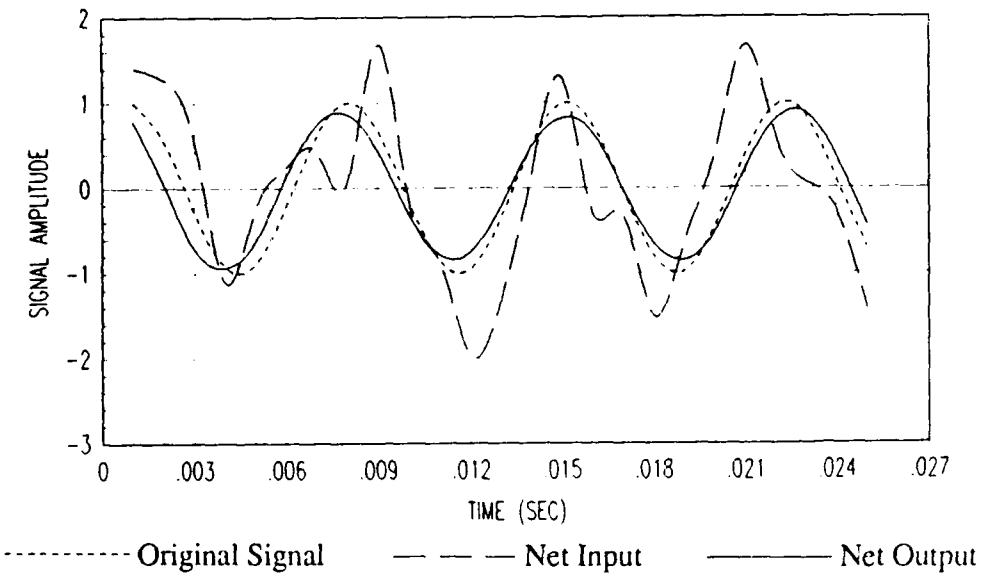

Figure 11. Test 1 - Network Behavior with 140 Hz Experimental Input Signal.

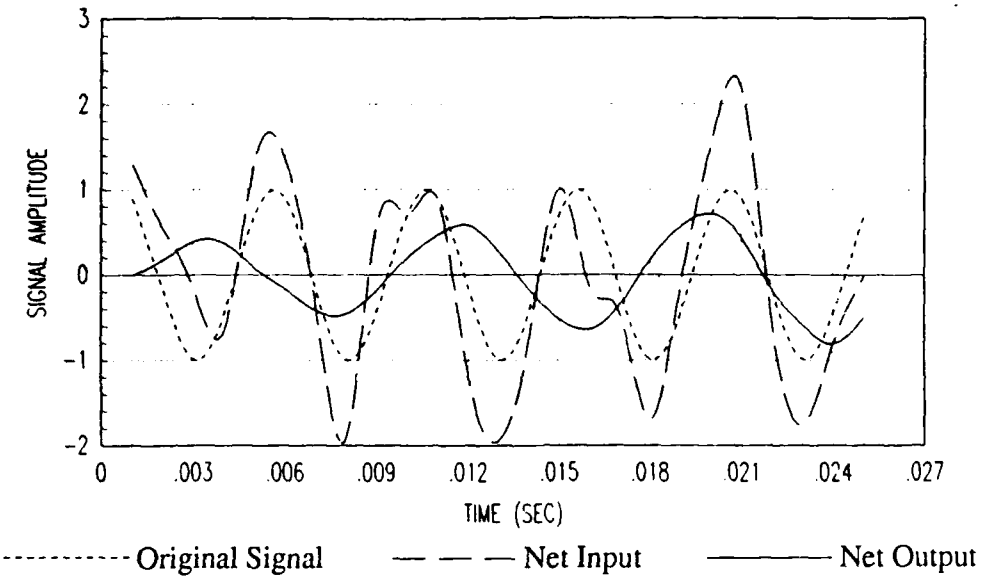

Figure 12. Test 1 - Network Behavior with 200 Hz Experimental Input Signal.

As can be seen from these graphs, the network performed as expected and did a better job of generating the appropriate output vectors for signals whose frequencies were closer to the training set. The utility of the graphs was not limited to just this analysis, however. A comparison of the graphs revealed that the network's output vectors all had one thing in common. They all had approximately three periods in the sampling window, even though the input signals had anywhere from one to five periods. This implied that the network was generating output vectors that were limited to a narrow frequency band, and that perhaps the network was unable to transmit frequencies other than those learned from the training set. In an effort to determine what was actually causing this phenomenon, FFT graphs of the original signal, of the signal corrupted by noise and used as a network input, and of the network's output were plotted. Some of these results are presented in Figures 13, 14, 15, 16, 17, 18, 19, 20, and 21.

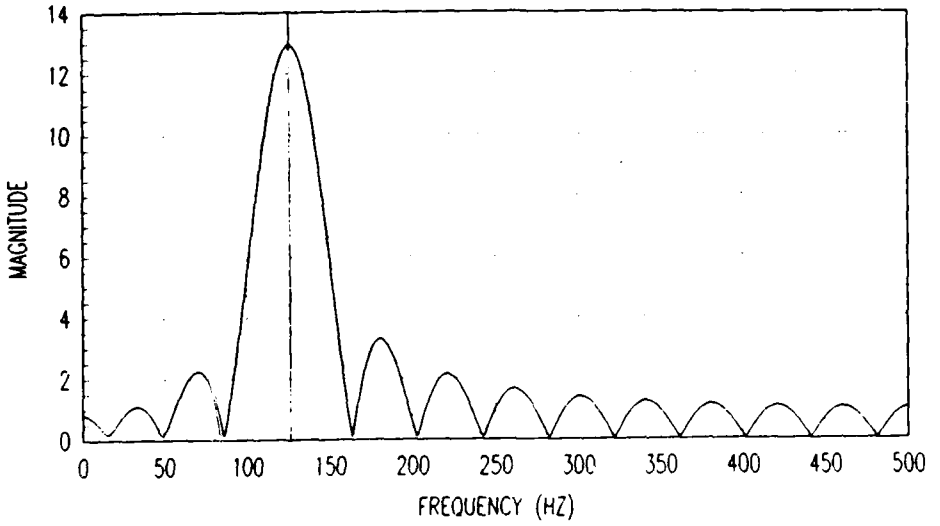

Figure 13. Test 1 - FFT of Original 124 Hz Signal.

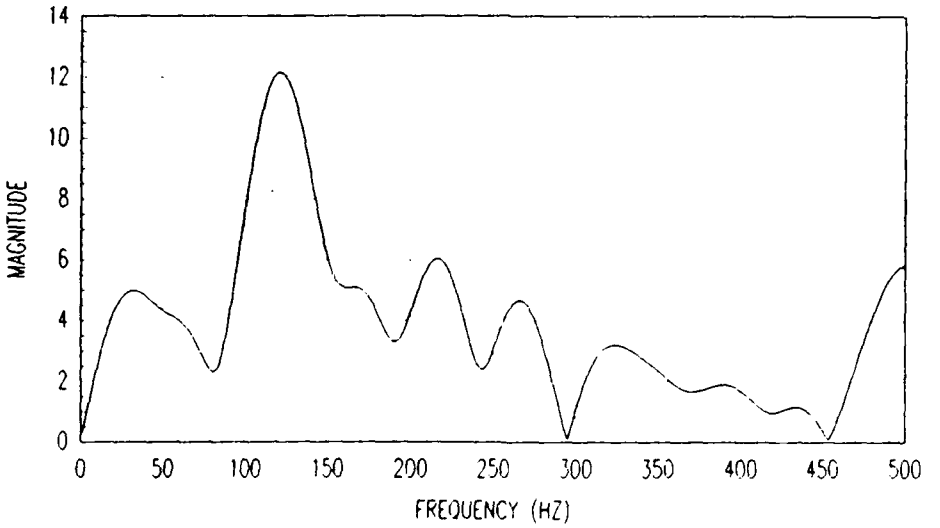

Figure 14. Test 1 - FFT of Noise Corrupted 124 Hz Input Signal.

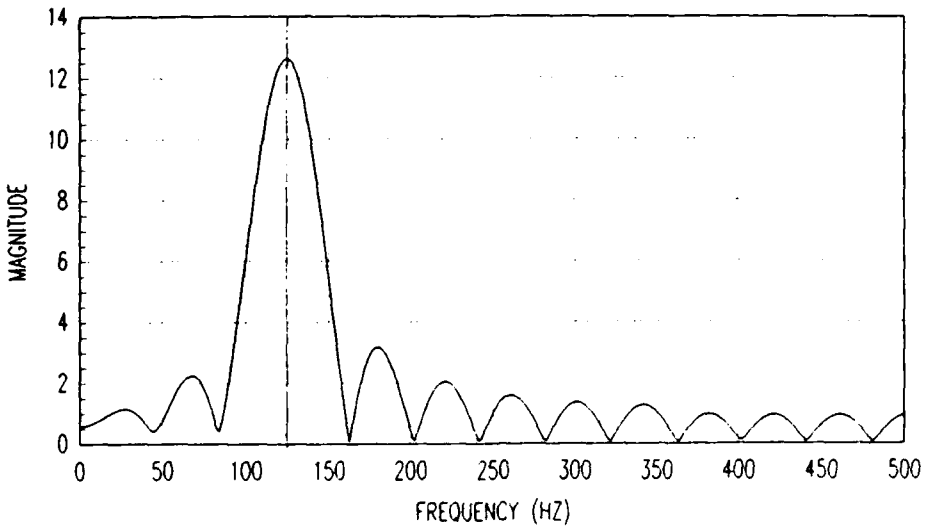

Figure 15. Test 1 - FFT of Network's Output Corresponding to 124 Hz Input.

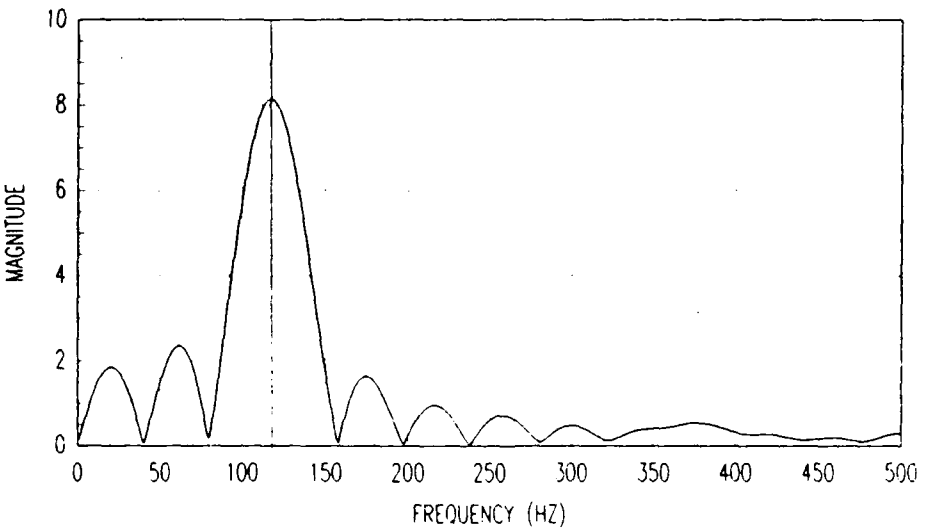

Figure 16. Test 1 - FFT of Network's Output Corresponding to 50 Hz Input.

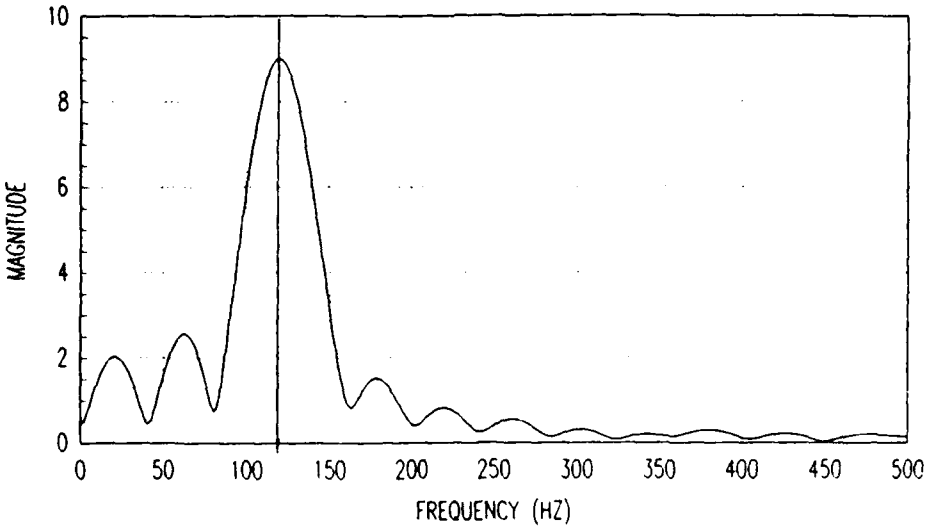

Figure 17. Test 1 - FFT of Network's Output Corresponding to 80 Hz Input.

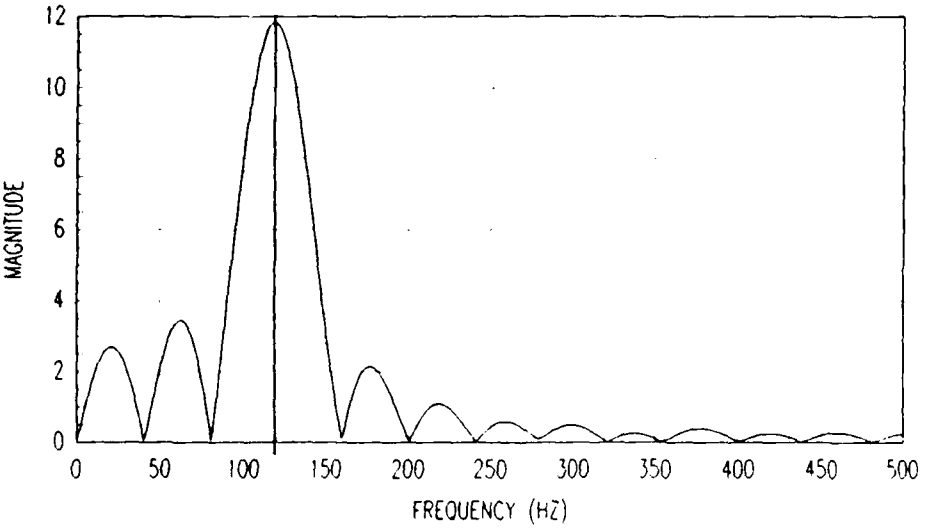

Figure 18. Test 1 - FFT of Network's Output Corresponding to 110 Hz Input.

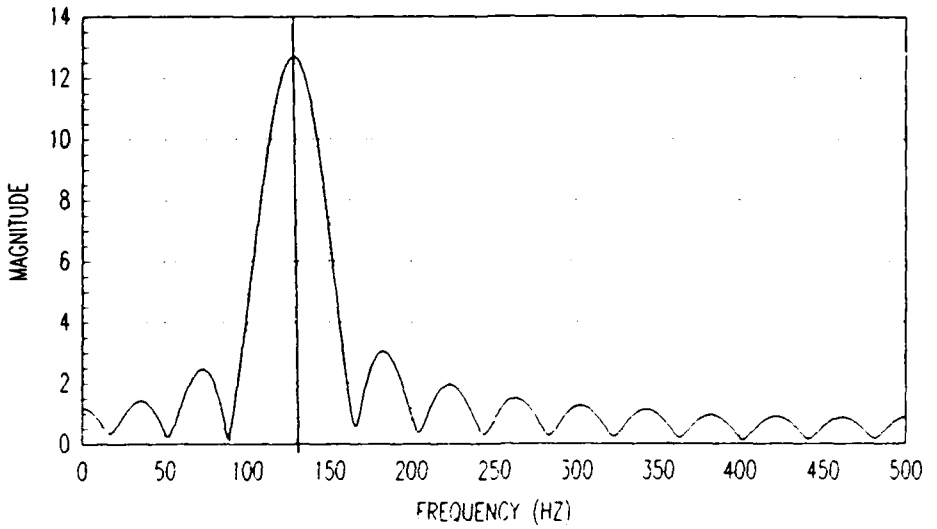

Figure 19. Test 1 - FFT of Network's Output Corresponding to 130 Hz Input.

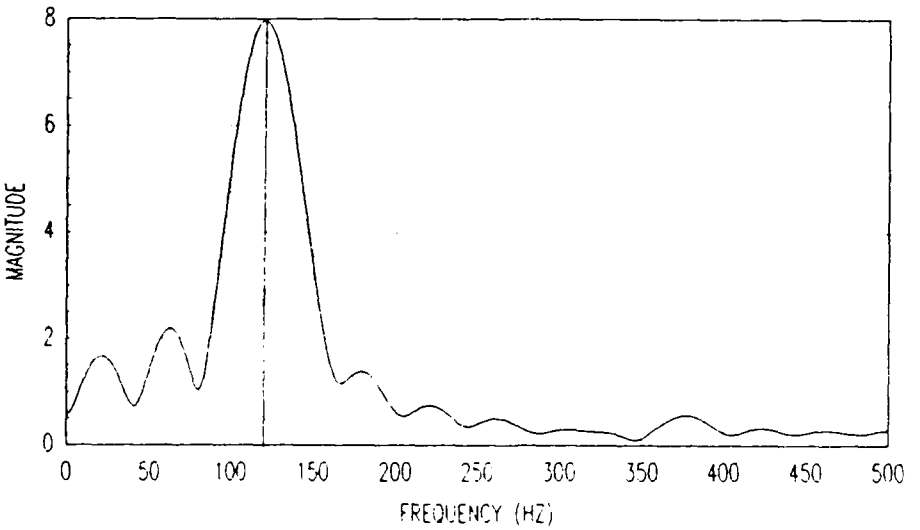

Figure 20. Test 1 - FFT of Network's Output Corresponding to 170 Hz Input.

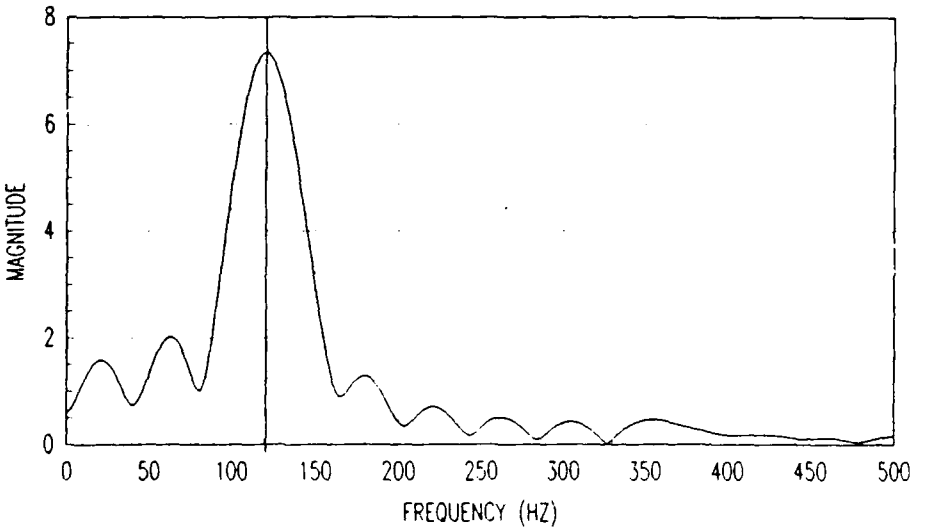

Figure 21. Test 1 - FFT of Network's Output Corresponding to 200 Hz Input.

An evaluation of the FFT graphs provided the desired information. The network had a natural frequency response of about 120 Hz and was able to shift its output frequency only slightly in an attempt to match the input. However, as the input signal's frequency moved further away from those of the training set, the network's performance quickly dropped off as the network appeared to be unable to shift its output frequency beyond the frequencies used in the training set.

This effect could also be noticed when the amplitude of the network's output signal was examined. For input frequencies that were close to the training frequencies, the maximum amplitude of the network's output was right around 1.0; but, as the frequency drifted away from the training range, the maximum amplitude began to drop. In an effort to generalize the network's performance over a broad range of frequencies, input signals which were uncorrupted by noise were provided to the network, and the amplitude of the network's output was plotted. As can be seen from Figure 22, the network's amplitude

remained around 1.0 for frequencies within 15 Hz of the original training frequencies. As the frequencies moved outside of this range, however, the amplitude began to drop off. Note that the curve in Figure 22 cannot be interpreted as a typical filter gain curve.

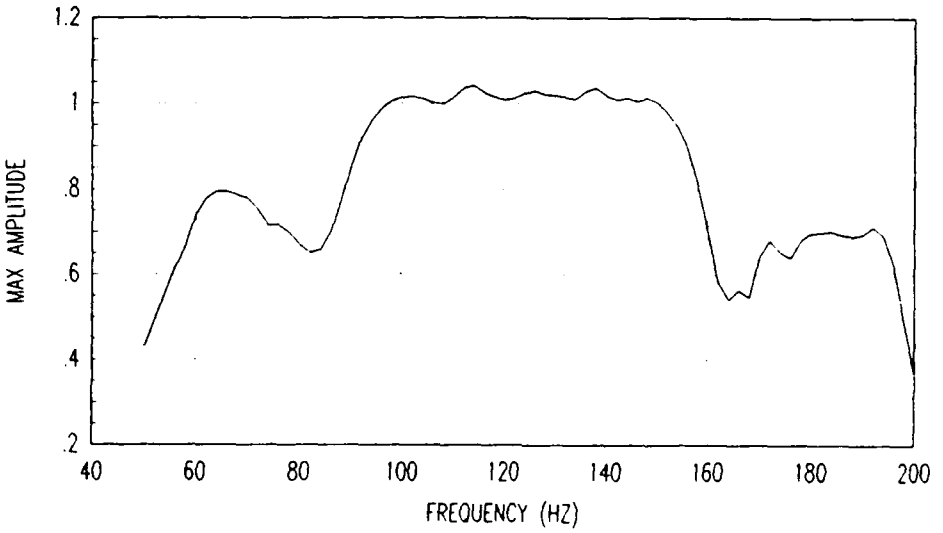

Figure 22. Test 1 - Network's Amplitude Response.

One last measurement was made in this experiment which helped to identify how the network was operating. When the impulse response of the network was calculated, an indication of how the network computed the phase was revealed. Five inputs vectors were generated and provided to the network. In each vector, all 25 elements were set to 0.0, except for one which was set to 1.0. Figures 23 and 24 showed that the peak amplitude of the network's output occurred at the time which corresponded to the impulse signal, and indicated that the network used the maximum input value to generate the correct phase for the output signal.

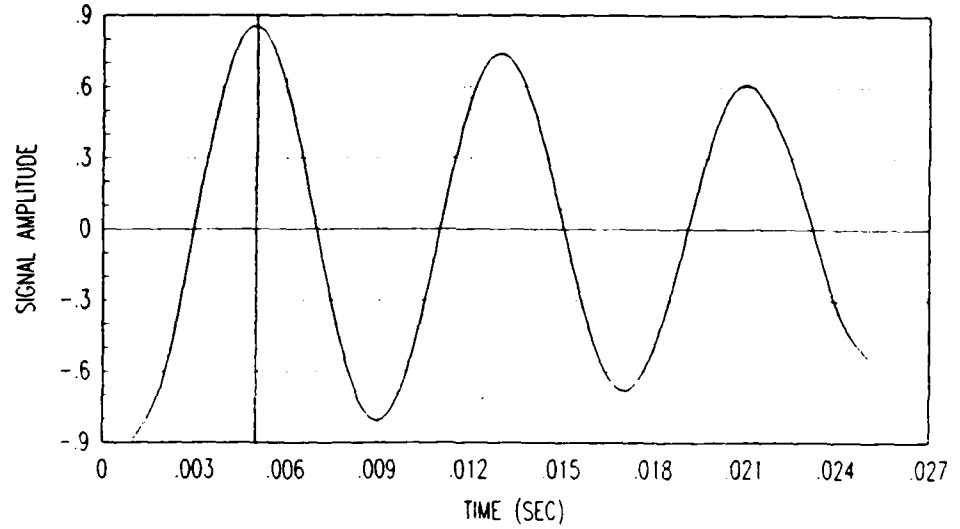

Figure 23. Test 1 - Network's Output for Impulse Occurring at 0.005 sec.

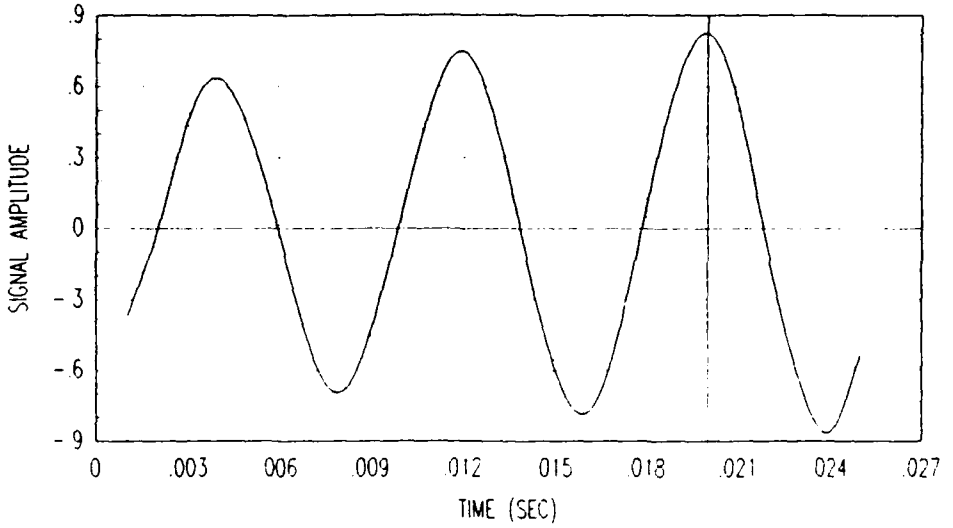

Figure 24. Test 1 - Network's Output for Impulse Occurring at 0.020 sec.

Test 2 - Narrow Frequency, Fixed Amplitude, Moderate Noise. In this experiment a lower signal to noise ratio was desired, so the noise vectors were generated with the same amount of power as the original signal. This meant that the noise vectors had to be generated from a Gaussian distribution with a variance of 0.5. The sine waves used for training had random frequencies, ranging from 120 Hz to 130 Hz, and fixed amplitudes of 1. All other aspects this experiment were identical to Test 1.

The performance measurements taken during training revealed that the network was not able to match the input and output signals quite as accurately as in Test 1; and that, just like Test 1, most of the training had been completed after 10,000 training samples, see Figures 25 and 26. Knowing this, the question which had to be answered was how this increased error manifested itself.

Examination of graphs like the ones presented for Test 1, showed that the network was more resistant to moving away from its natural frequency. The network was still able to adjust its output frequency, but was only willing to do so when the frequency of the input signal was further away from the natural frequency than in Test 1. Even then, the network only made a small adjustment. In addition, the network seemed unable to compute the phase as accurately as before, and caused the output signal to lag behind the input signal, see Figure 27. One additional change in network behavior was noted. The network's amplitude response dropped off earlier than in Test 1, and the drop was more pronounced, see Figure 28.

All of this indicated that the network "knew" what the output signal was supposed to be, but was having difficulty getting the information required to generate it. This interpretation was reasonable, since there was four times the amount of noise in Test 2 than in Test 1. The additional noise made finding the exact signal more difficult, and the output signals became less accurate approximations of the input vector. This is what was believed to have caused the performance degradation.

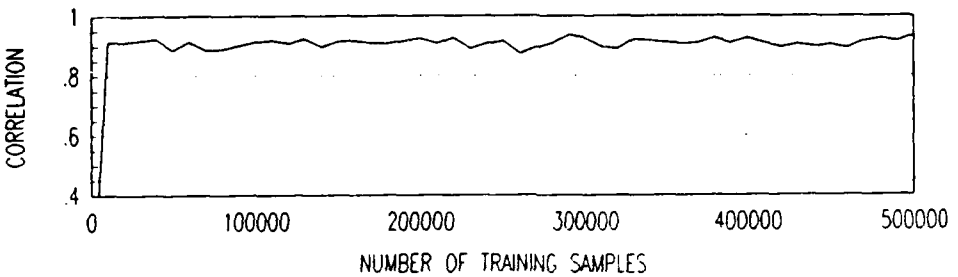

Figure 25. Test 2 - Signal Correlation During Training.

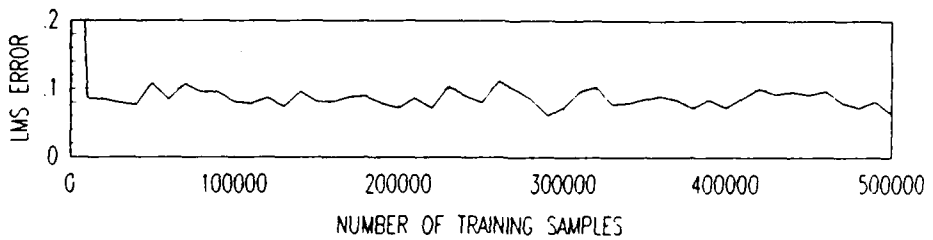

Figure 26. Test 2 - Signal Mean Square Error During Training.

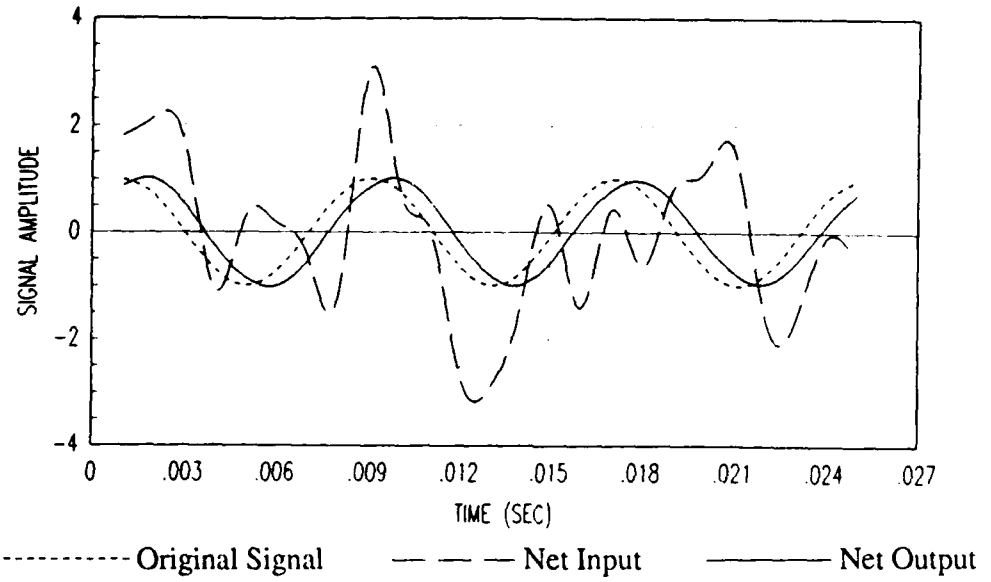

Figure 27. Test 2 - Network Behavior with 124 Hz Experimental Input Signal.

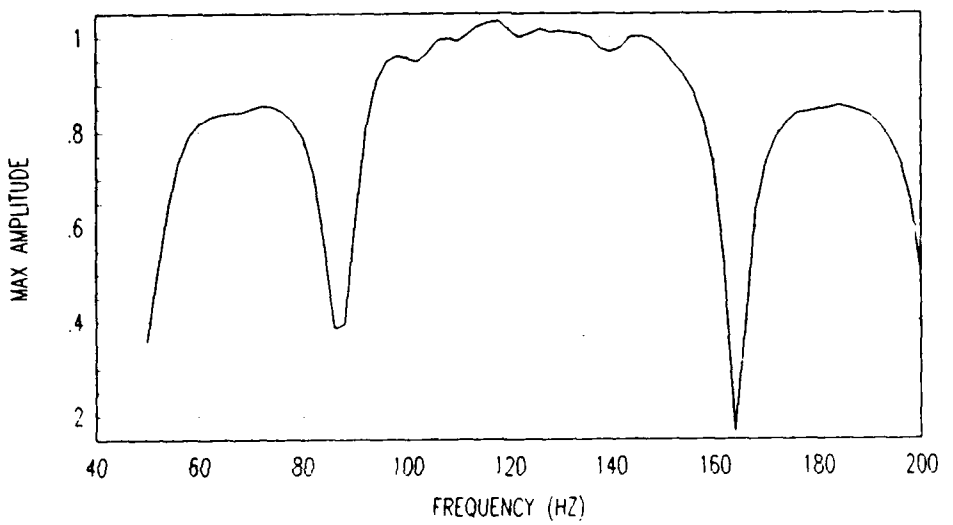

Figure 28. Test 2 - Network's Amplitude Response.

Test 3 - Broad Frequency, Fixed Amplitude, Moderate Noise. In an attempt to determine how a network would respond when it was trained with a broader set of frequencies, the sine waves used in this experiment had random frequencies which ranged from 110 Hz to 140 Hz. This was three times the frequency band used in the previous experiments. Signals still had fixed amplitudes of one, so the power associated with each signal remained at 0.5. In addition, the same signal to noise ratio as in Test 2 was desired, so the noise vectors were generated from a Gaussian distribution with a variance of 0.5.

This experiment produced another drop in the performance measurements taken during training, see Figures 29 and 30. Although the network had some problems computing the phase and frequency of the input signal, the increased error experienced in this experiment was not caused by more problems with these computations. Instead, the increased error seemed to be caused by a measurable amount of harmonic distortion. The error occurred in both the second and the third harmonics, and increased as the frequency used for training approached the boundaries of 110 Hz and 140 Hz. For training frequencies which were close to 125 Hz, the distortion was almost non-existent. Figures 31, 32, 33, and 34 show the changes in harmonic distortion that occurred when the input frequency was altered.

One other difference was noted between the performance of the network in this experiment and the performance of the networks in the previous experiments. The difference was that the network's amplitude response became much more erratic, see Figure 35. The primitive shape of the previous curves remained intact, but the smooth continuity suffered across the frequencies tested. Unfortunately, after a great deal of analysis and testing, no explanation could be made for either the harmonic distortion or the frequency response degradation.

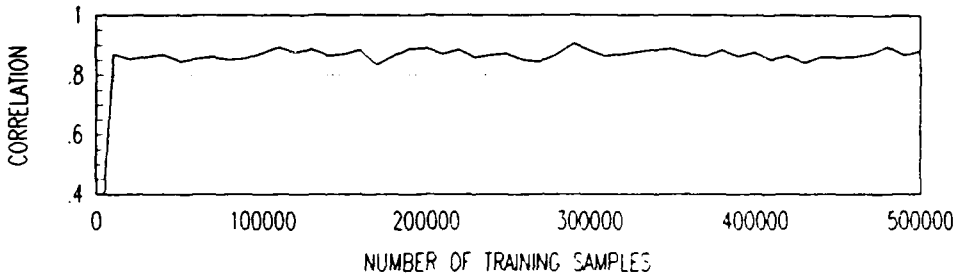

Figure 29. Test 3 - Signal Correlation During Training.

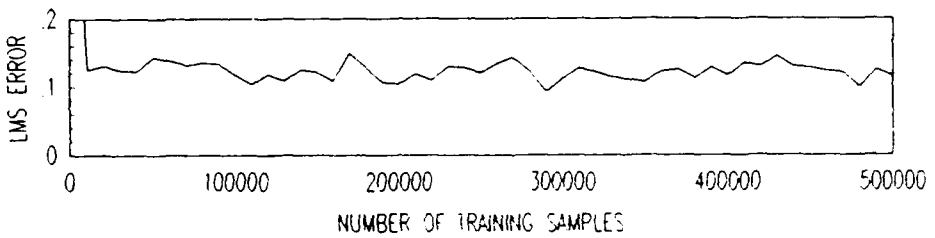

Figure 30. Test 3 - Signal Mean Square Error During Training.

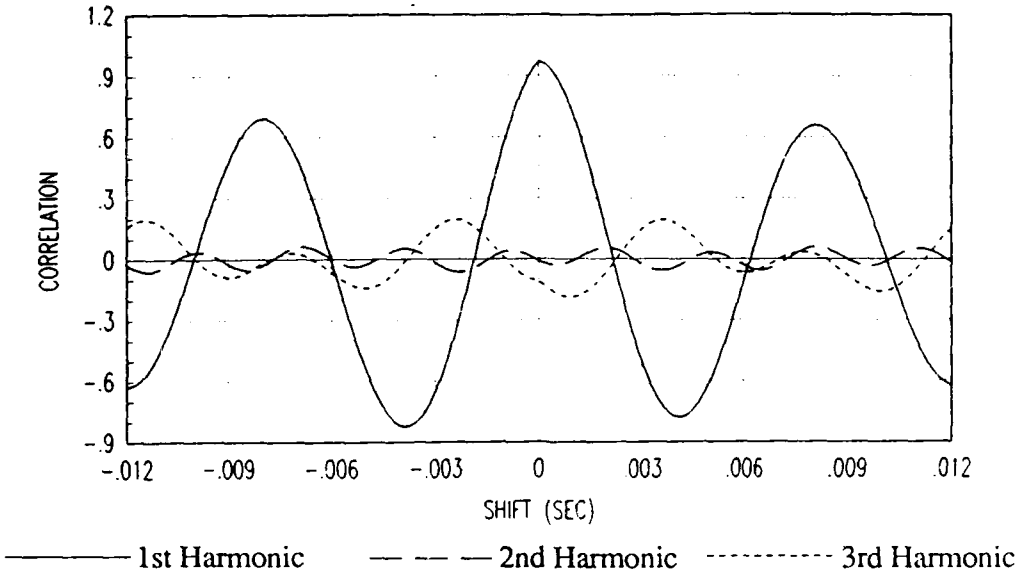

Figure 31. Test 3 - Harmonic Distortion with 110 Hz Experimental Input Signal.

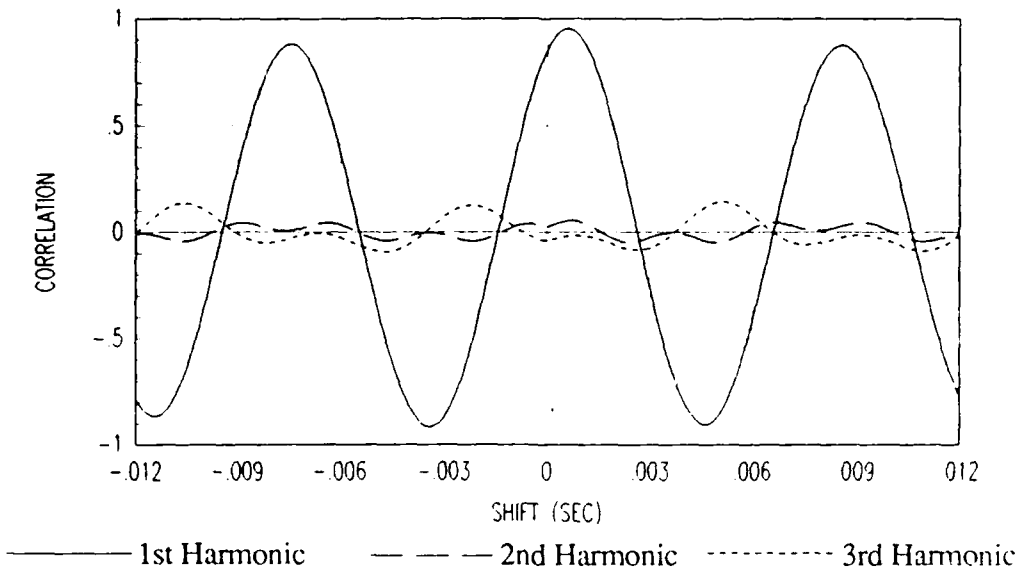

Figure 32. Test 3 - Harmonic Distortion with 120 Hz Experimental Input Signal.

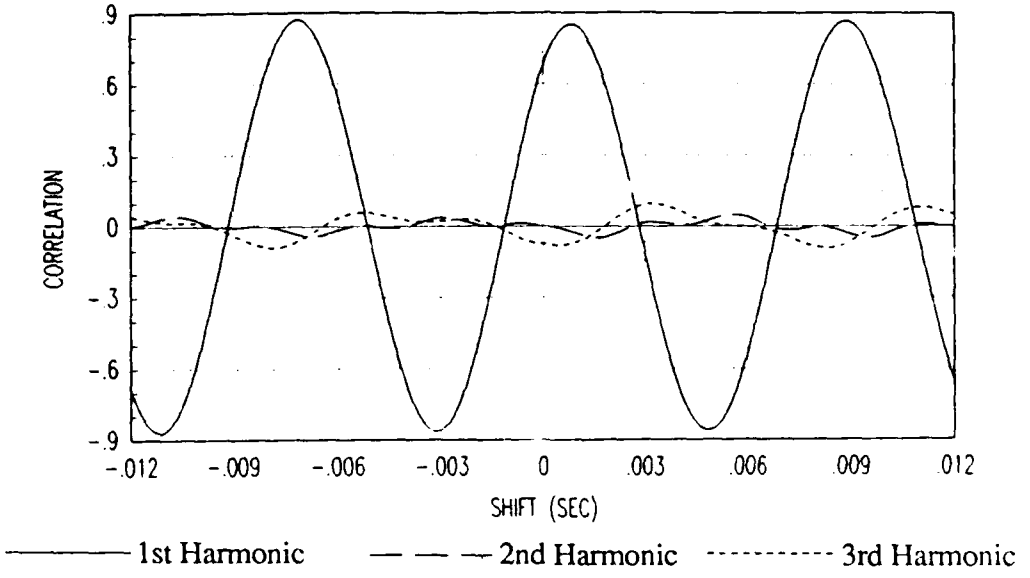

Figure 33. Test 3 - Harmonic Distortion with 130 Hz Experimental Input Signal.

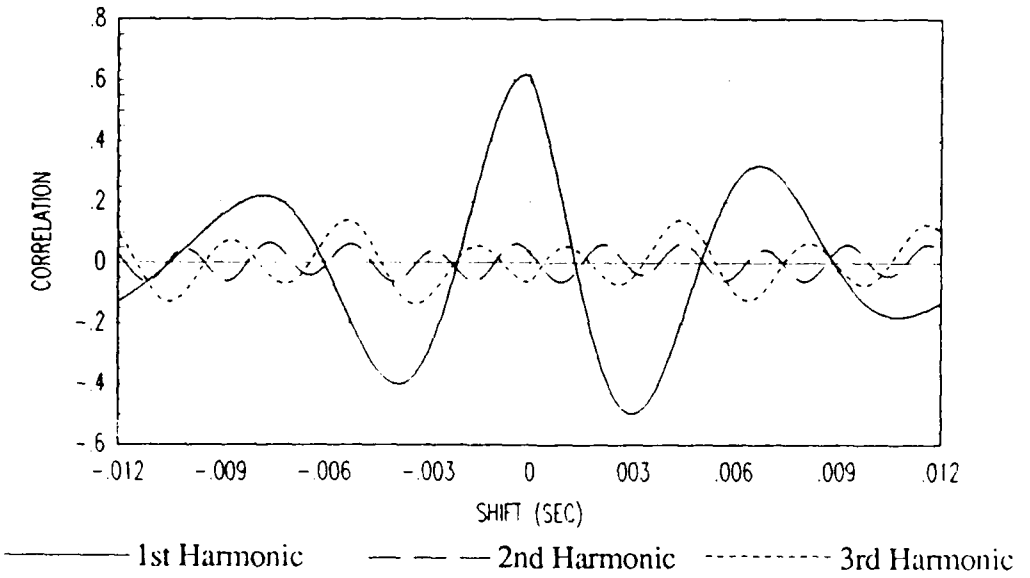

Figure 34. Test 3 - Harmonic Distortion with 140 Hz Experimental Input Signal.

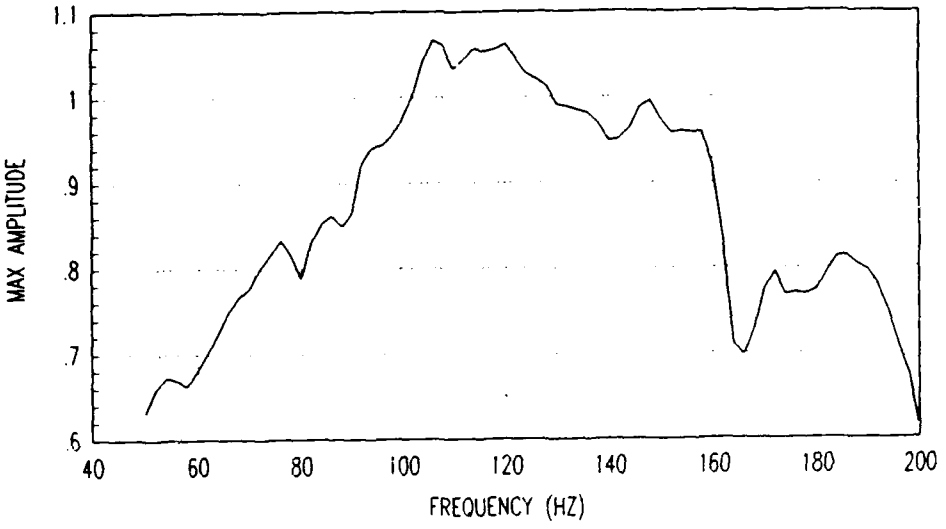

Figure 35. Test 3 - Network's Amplitude Response.

Test 4 - Broad Frequency, Fixed Amplitude, High Noise. This experiment was designed to increase the amount of noise corruption in the neural network's input signal. Each sine wave generated for this experiment was corrupted by noise with 10 times the power of the original signal. This meant that the noise vectors were to be generated from Gaussian distributions with a variance of 5.0. The sine waves were generated using the same parameters as in Test 3.

As can be seen from Figures 36 and 37, the performance measurements taken during training indicate that the network's ability to create the correct output was significantly diminished. This could have been expected since spotting trends in a signal that had an order of magnitude more noise than signal would have been difficult. After reviewing the signal and FFT graphs, the conclusion was that the network was unable to determine what the input signal had been, and was generating an output signal using a constant frequency of 120 Hz. This is evidenced in Figures 38 and 39 where the output spectrum seemed to be independent of the frequency of the input signal.

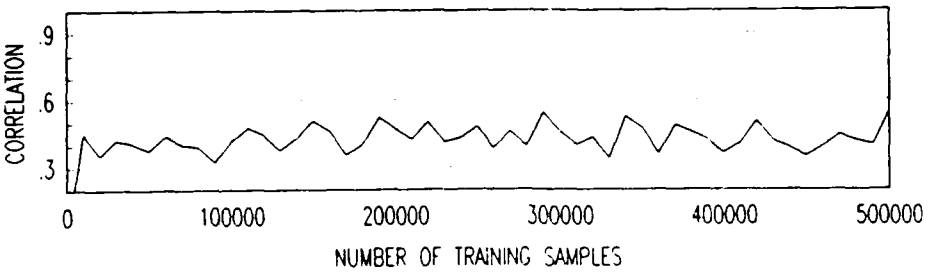

Figure 36. Test 4 - Signal Correlation During Training.

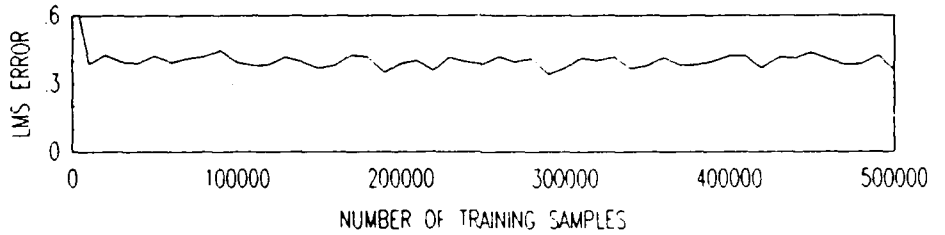

Figure 37. Test 4 - Signal Mean Square Error During Training.

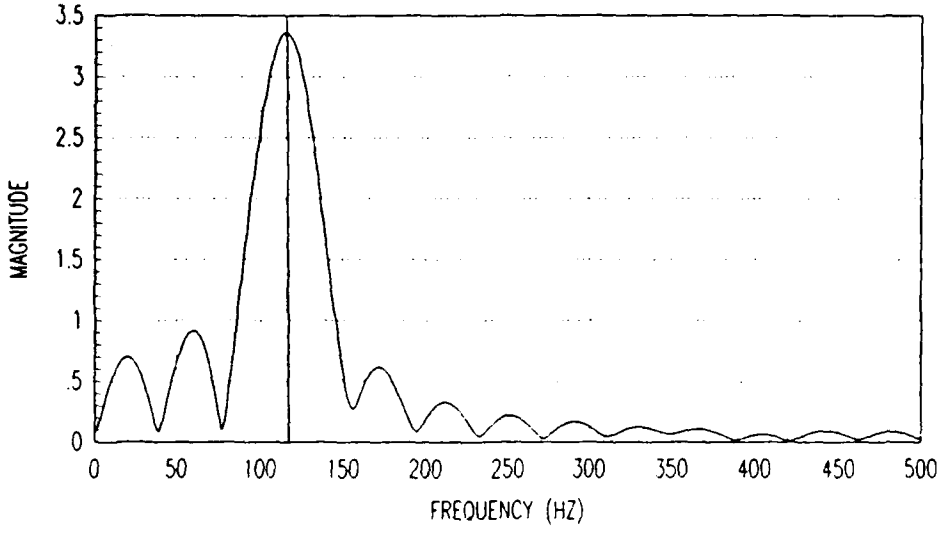

Figure 38. Test 4 - FFT of Network's Output Corresponding to 110 Hz Input.

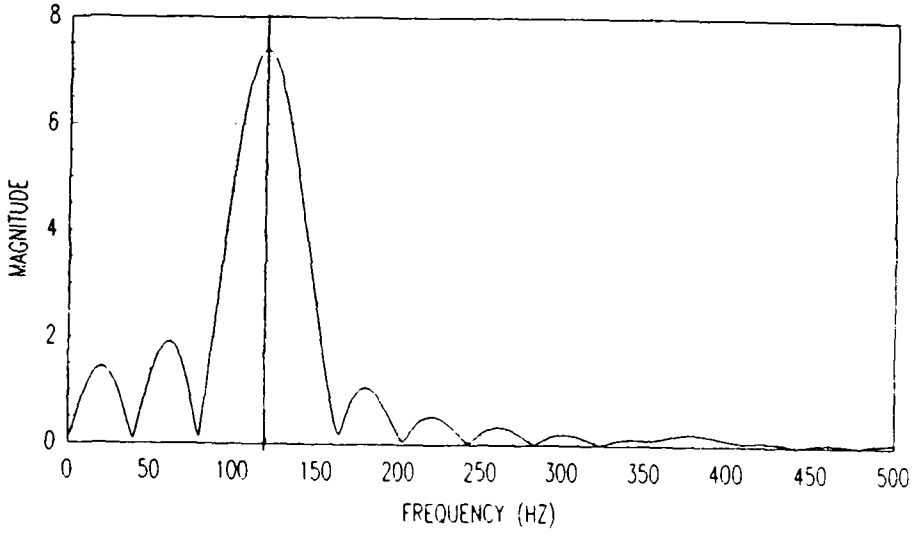

Figure 39. Test 4 - FFT of Network's Output Corresponding to 140 Hz Input.

Test 5 - Broad Frequency, Random Amplitude, Moderate Noise.

Human speech can be approximated by summing many sine waves of various frequencies and amplitudes. The previous experiments began to show how a neural network may be able to filter noise from a single sine wave of fixed frequency. This experiment broadens that knowledge by investigating single sine waves of random amplitudes. Each sine wave generated for this experiment had its amplitude scaled by a random value from the uniform distribution [0,1]. The same signal to noise ratio as in Test 3 was desired, but the power of the sines waves used in this experiment was not known apriori. To avoid altering more than one variable, the noise vectors continued to be generated from a Gaussian distribution with a variance of 0.5.

The performance measurements indicated that the network had a great deal of trouble generating the appropriate output signals, see Figures 40 and 41. The correlation factor dropped from 0.87 in Test 3 down to about 0.58 in this experiment. Finding the causes of this drop was difficult, and yielded only a few possibilities.

To find out how the network was behaving, three groups of test signals were used after training had been completed. The first set generated sine waves with amplitudes of 0.33, corrupted them with noise, and provided them as inputs for the network; the second set used amplitudes of 0.66; and the third set used amplitudes of 1.0. The outputs generated by the network were collected and analyzed using the same techniques as in the previous experiments.

Examining the signal plots revealed that the network was able to generate a signal which generally approximated the original, see Figures 42, 43, and 44. One important item was noted at this point. The network's output seemed to have an amplitude of about 0.4, regardless of the original signal size. This indicated that the network was unable to compute how much energy had been in the original signal, and that it was using an output amplitude that did the best job of matching the inputs seen in the training set.

The FFT plots were able to disclose a couple of other things. First, as the amplitude of the original signal diminished, the network tended to generate an output signal of about 115 Hz, the network's natural frequency. This implied that as the amplitude dropped, the network found it impossible to determine the frequency and used a generic output signal. The frequency of this generic output was the network's best attempt at generating a signal which was as close as possible to all the outputs seen in the training set. Second, as the amplitude of the original signal increased, the network became more accurate at computing the frequency and shifted the frequency of the output to match that of the original signal, see Figures 45, 46, and 47.

All of this indicated that there was a noise threshold critical to the network. For signal to noise ratios below the threshold, the network was unable to find the underlying fundamental frequency and generated a generic output signal. However, for signal to noise ratios above the threshold, the network was able to find the frequency and generated an output which approximated the original signal.

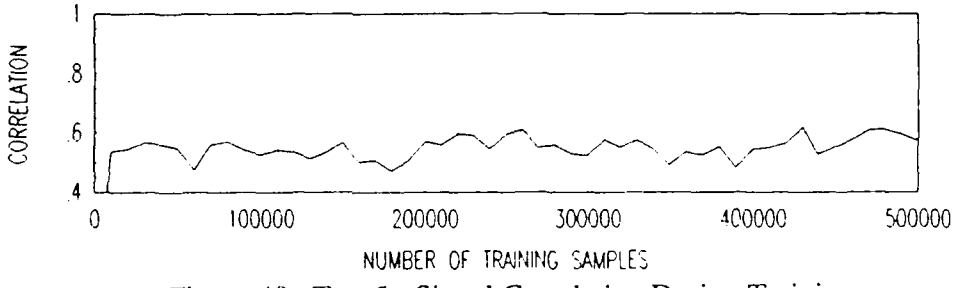

Figure 40. Test 5 - Signal Correlation During Training.

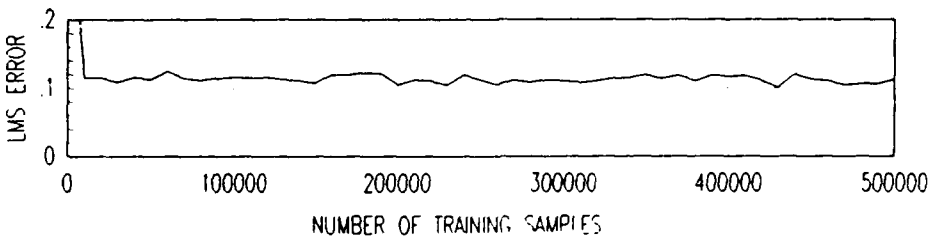

Figure 41. Test 5 - Signal Mean Square Error During Training.

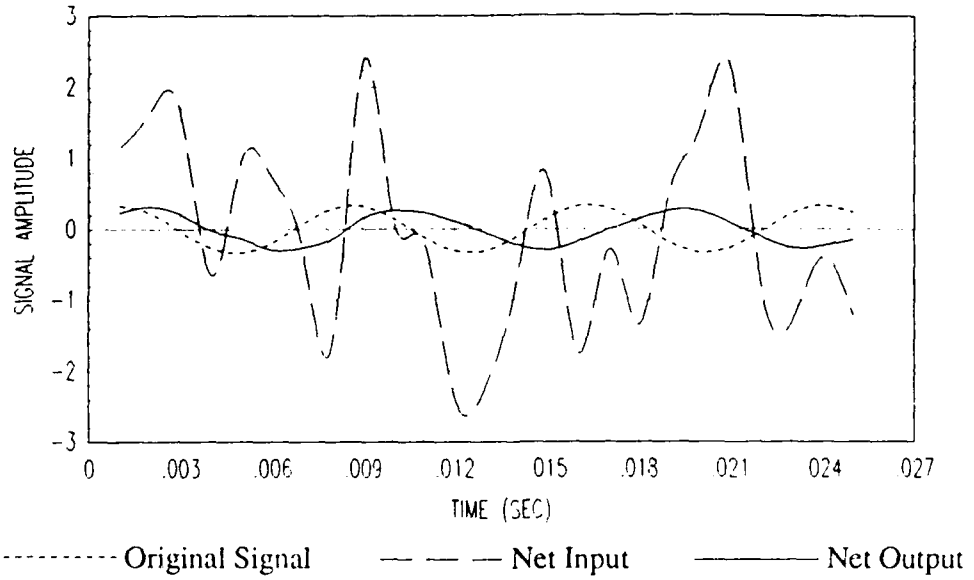

Figure 42. Test 5 - Network Behavior with 130 Hz / 0.33 Amplitude Input Signal.

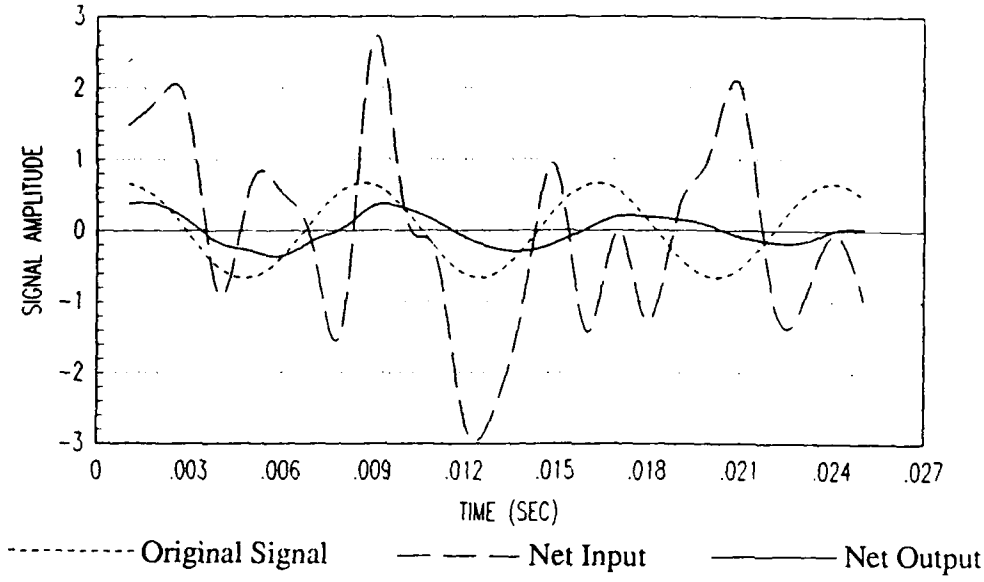

Figure 43. Test 5 - Network Behavior with 130 Hz / 0.66 Amplitude Input Signal.

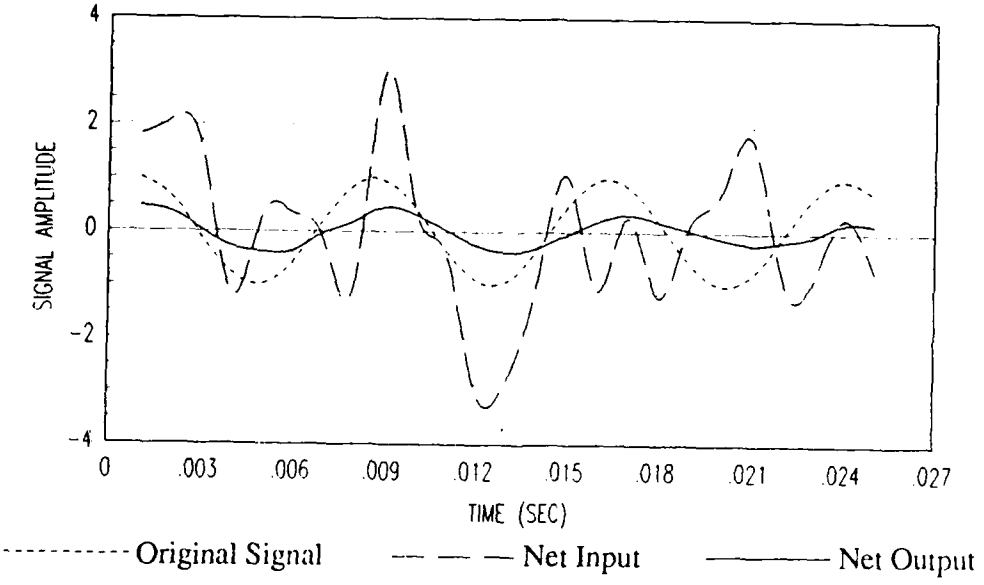

Figure 44. Test 5 - Network Behavior with 130 Hz / 1.0 Amplitude Input Signal.

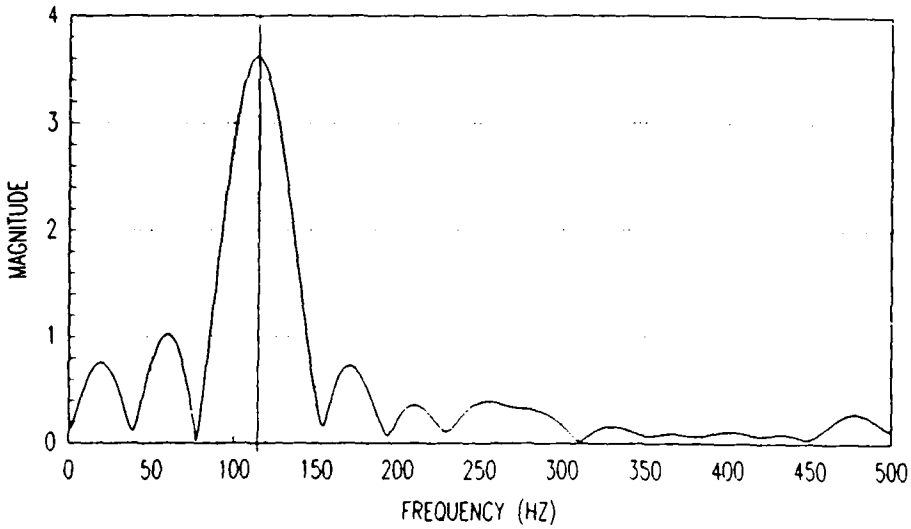

Figure 45. Test 5 - FFT of Network's Output with 130 Hz / 0.33 Amplitude Input.

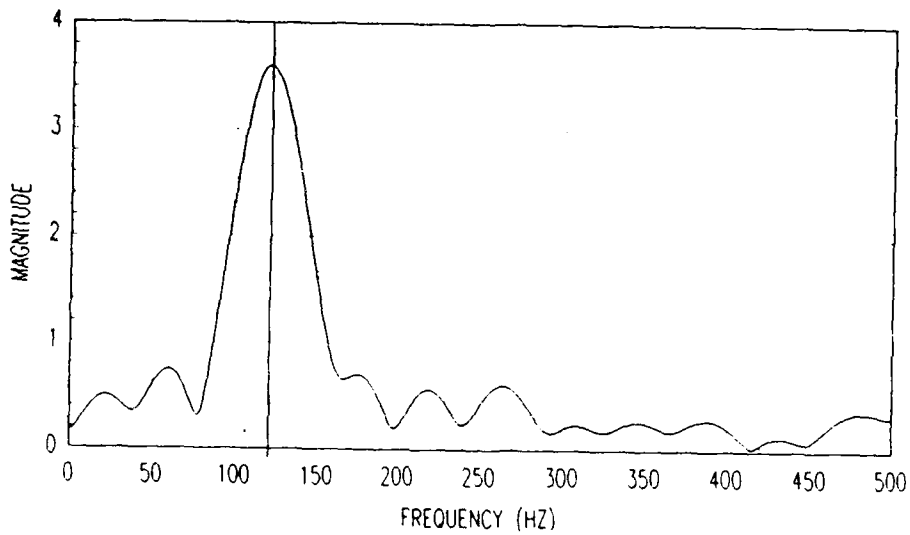

Figure 46. Test 5 - FFT of Network's Output with 130 Hz / 0.66 Amplitude Input.

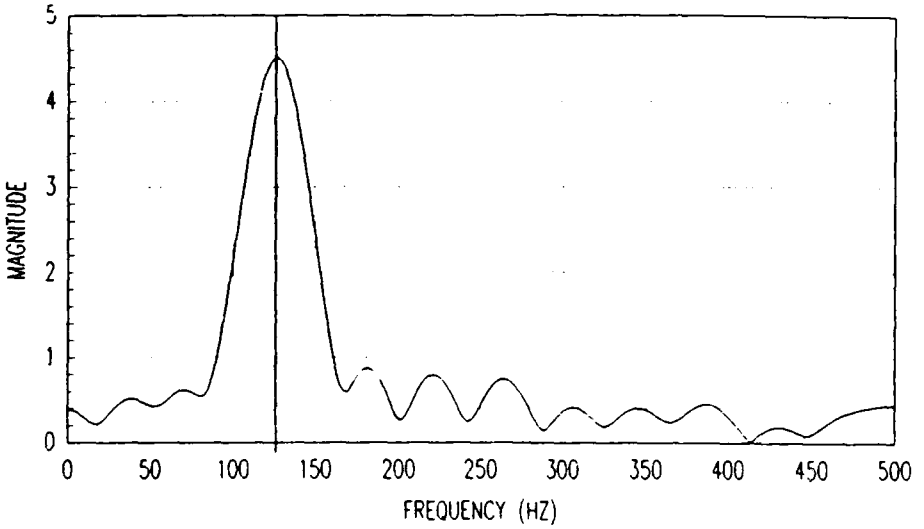

Figure 47. Test 5 - FFT of Network's Output with 130 Hz / 1.0 Amplitude Input.

Multiple Sine Wave Noise Reduction

In this second group of tests, the objective was to take a step towards more complex and realistic problems, while still being able to build upon the hypotheses and results of the first group. To achieve this goal, multiple sine waves were generated and summed together to create the input signal. This technique provided the benefit of knowing the discrete frequencies used to generate the input , and of being able to use this information to evaluate the accuracy of the network. On the other side, the network had to learn how to filter noise from an input that was much more complex than a single sine wave.

Approach and Assumptions. The nets used in this sequence of experiments still had twenty-five nodes in each layer. Again, the justification was that the speech data used by Tamura and Waibel was more complex than the multiple sine wave inputs used

here, so the ability of the network to filter noise was not expected to be impacted by the smaller number of nodes compared to the 60 used by Tamura and Waibel.

The linear scaling technique used for this group of experiments was different, however. Each single sine wave produced maximum values of +1 so that the largest amplitude which could have been generated by three waves was +3; the minimum possible amplitude was -3. In order to scale these signal so that its amplitude did not exceed the range [0,1], the signal was shifted by adding 4.5, and then scaled by dividing by 9. The range of possible input values produced using this scaling technique was 0.17 to 0.83; exactly the same range of input values produced when the single sine waves were scaled. By using the same range of values, changes in network performance could be compared with the single sine wave results to determine how filtering a more complex signal affected the network.

The sine waves produced for these experiments had random phase uniformly distributed over the range 0^{o} to 360^{o}. Using a random phase ensured that the network did not "learn" to generate a complex signal made up of sine waves that had a fixed time relationship. It also helped to ensure that the sine waves created a unique complex signal for every training input. Each sine wave had, in addition to its own phase, its own frequency band. The first sine wave was randomly distributed over a range from 10 Hz to 40 Hz, the second was over the range 110 Hz to 140 Hz, and the third was over the range 210 Hz to 240 Hz. These values were selected to allow the signals to have frequencies which were multiples of each other, as well as ensuring there were cases where the frequencies were totally unrelated.

One final issue that had to be addressed was computing the power of the complex signals. The previous equation for computing power was only useful in computing the power of a sinusoidal signal. However, Parseval's Theorem states that the average power of a complex periodic signal can use the same formula. Since the waveforms in

this experiment were of random phase, and since there were a large number of samples being generated, the power of the signal is the sum of the individual signals' power. This meant that the power could be computed with the same formula used previously [Kabrisky, 1988b]. Knowing this allowed the signal to noise ratios to be calculated and the results to be compared with those obtained in the single sine wave experiments.

Test 6 - Broad Frequency, Fixed Amplitude, High Noise. This first experiment was designed specifically to use a low signal to noise ratio; a ratio of 1:3. Three sine waves with amplitudes of 1 were used to generate the complex input signal and this meant that the power associated with the original signal was 1.5. Therefore the noise vector had to be generated from a Gaussian distribution with a variance of 4.5.

Unfortunately, the results of this test indicated that the network was not able to filter the noise nearly as well had been done with single sine waves. Examination of the performance measurements showed that the correlation factor had reached only about 0.50 during training, roughly the performance achieved when filtering a single sine wave that had a signal to noise ratio of 1:10. Although most of the training was completed after 10,000 training samples, the network continued to learn until training stopped, see Figures 48 and 49.

Review of the signal graphs indicated that the network was only able to generate an approximation of the original signal, see Figures 50 and 51. Insight to what the network was doing came after examining the FFT of the output signal. Figures 52 and 53 show that regardless of the input signal, the network was generating an output signal which had three main frequency components: 28 Hz, 120 Hz, and 230 Hz. Although the magnitude of these components varied, their frequencies moved very little and did not seem to be influenced by the input. The conclusion was that the network had found a general purpose output that provided the best match for the allowable range of inputs.

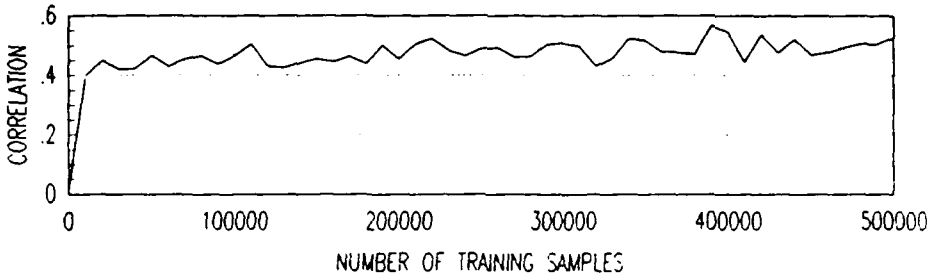

Figure 48. Test 6 - Signal Correlation During Training.

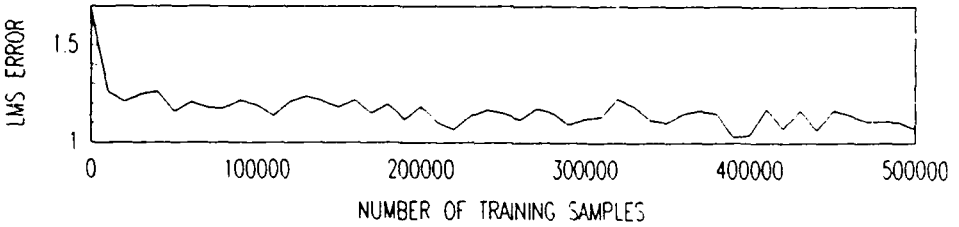

Figure 49. Test 6 - Signal Mean Square Error During Training.

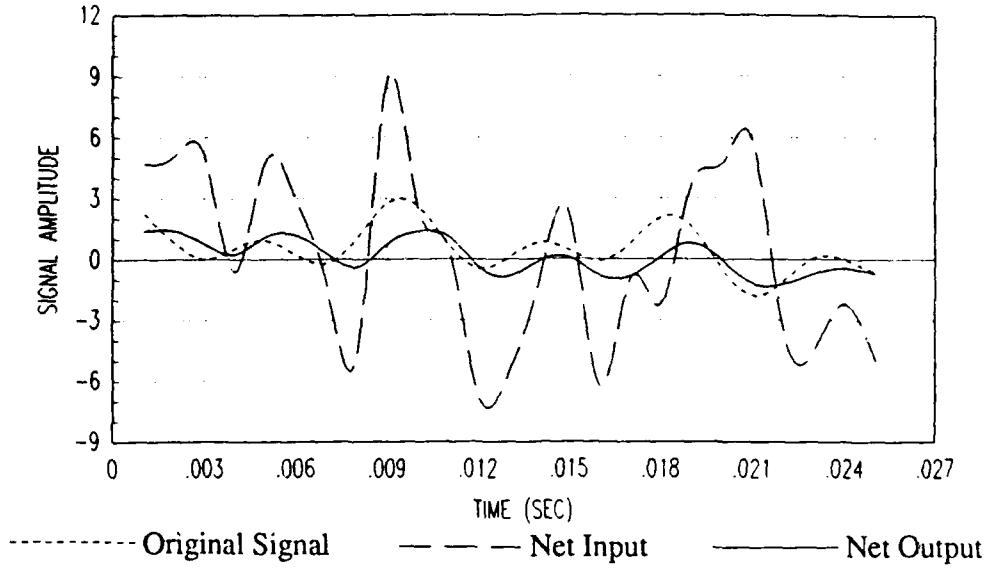

Figure 50. Test 6 - Network Behavior with 20 Hz / 120 Hz / 220 Hz Input Signal.

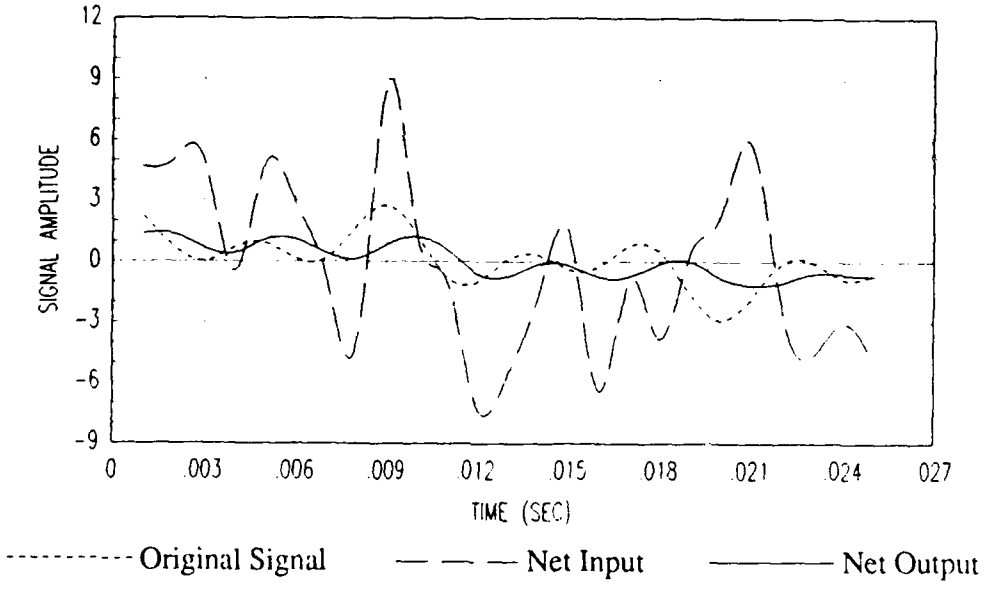

Figure 51. Test 6 - Network Behavior with 30 Hz / 130 Hz / 230 Hz Input Signal.

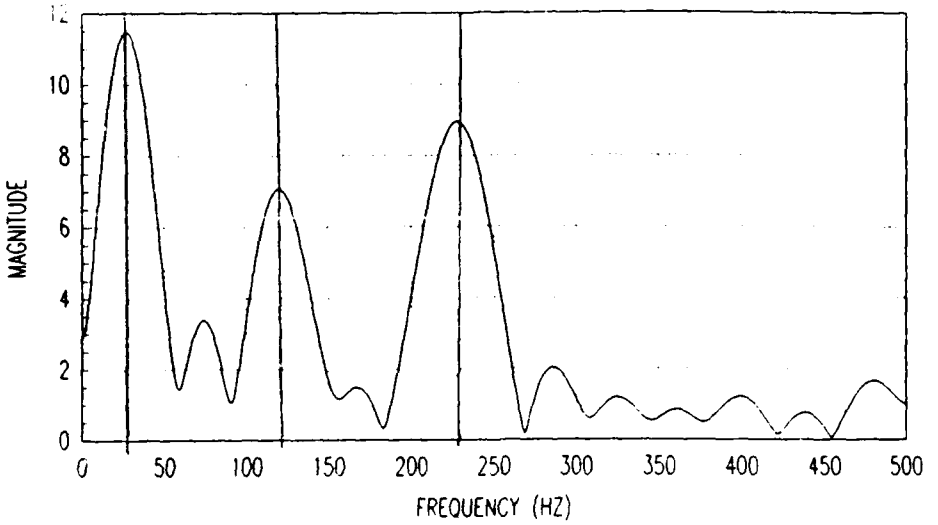

Figure 52. Test 6 - FFT of Network's Output with 20 Hz / 120 Hz / 220 Hz Input.

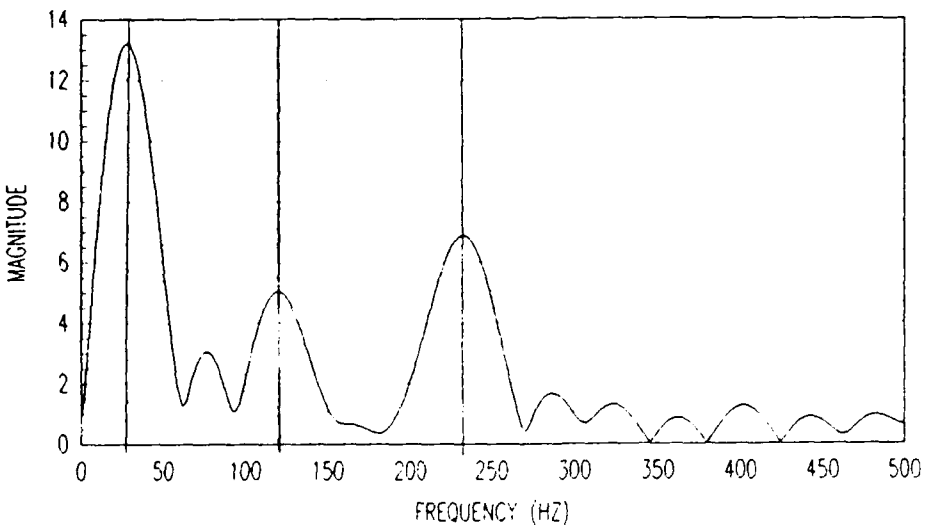

Figure 53. Test 6 - FFT of Network's Output with 30 Hz / 130 Hz / 230 Hz Input.

Test 7 - Broad Frequency, Fixed Amplitude, Extremely High Noise.

This particular experiment was designed to have an extremely low signal to noise ratio. From the previous experiment, it was known that the power of the original signal was 1.5. To obtain the target ratio of 1:30, the noise vector had to be generated from a Gaussian distribution with a variance of 45.0.

The resulting correlation measurements dropped to 0.05, much lower than in any previous experiment, see Figure 54. Mean square error measurements, however, showed that some sort of training had taken place, see Figure 55. A drop in performance had been expected, but the correlation factor of this experiment indicated that the network was unable to produce an output signal that resembled the input in any way. Such a low correlation factor suggested that the results of this experiment had little value, and that no useful information could be obtained by further examination. The sharp contrast in correlation factors between the multiple sine wave tests conducted to this point, and the single sine wave experiments, implied that the network had much more difficulty filtering noise from complex signals than it had with single sine waves.

However, an important assumption had been made earlier which may have affected the network's ability to filter these complex signals. The network had been constructed with 25 nodes on a layer instead of 60, and drastically reduced the input information the network was given. Time constraints prevented this possibility from being fully tested as part of the multiple sine wave experiments. However, since several networks with 60 nodes on a layer were to be trained to filter speech, some tentative conclusions could be reached. If the network's architecture had an inadequate number of nodes in this set of experiments, then when filtering speech the network's performance would be substantially better. However, if the network was actually unable to perform as a filter, then results similar to these were expected when filtering speech.

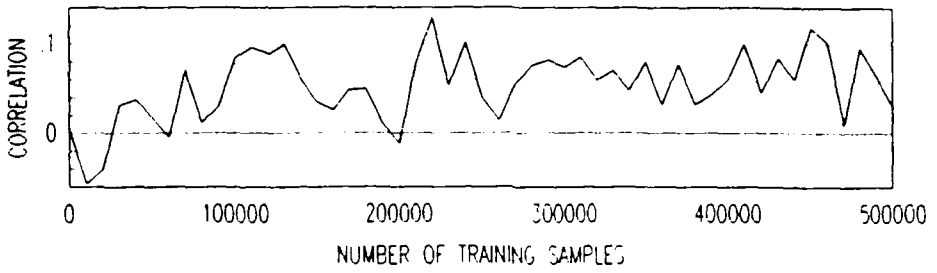

Figure 54. Test 7 - Signal Correlation During Training.

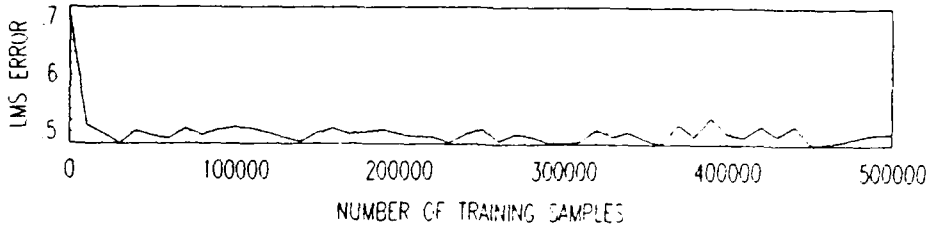

Figure 55. Test 7 - Signal Mean Square Error During Training.

Test 8 - Broad Frequency, Random Amplitude, Moderate Noise. This experiment was initiated before the results of Tests 6 and 7 were known. Consequently, the results that were obtained have little value other than to support the previous findings, and are presented here for completeness. This particular experiment was designed to show how a network behaved when it was given complex inputs with random amplitudes. For comparison purposes, this test was designed to have the same signal to noise ratio as in Test 6, a ratio of 1:3. However, since the power of the input signals was not known apriori, the noise vectors were generated from a Gaussian distribution with a constant variance. The variance was 4.5, the same variance used in Test 6.

The performance measurements for this experiment ended with a correlation value around 0.22 for training, see Figures 56 and 57. Although this is significantly higher than Test 7, the results are not spectacular and imply that the results should be reviewed skeptically. It is important to note, however, that the majority of training during this test did not occur within the first 10,000 training sets. Most of the training appeared to occur during the first 30,000 samples. This is a significant deviation from the performance measurements examined in the previous experiments.

Analysis of the FFT graphs provided little more. Regardless of the input frequencies and original signal amplitude, the output of the network remained basically the same. The maximum amplitude seemed to be about 0.4, the same amplitude as was seen in Test 5. This supports the hypothesis that for signals of varying amplitudes, the network picks an output amplitude that best matches the ones seen in the training set. The output frequencies were fixed as well. Figure 58 shows the three frequencies that made up the output signal: 25 Hz, 115 Hz, and 220 Hz. These values corresponded favorably with the values seen when a network was trained to filter a moderate amount of noise from a complex signal.

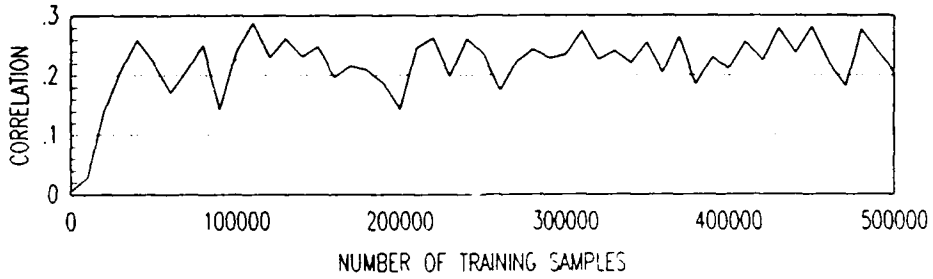

Figure 56. Test 8 - Signal Correlation During Training.

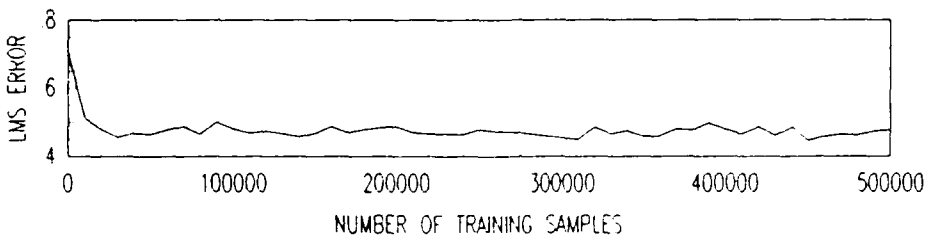

Figure 57. Test 8 - Signal Mean Square Error During Training.

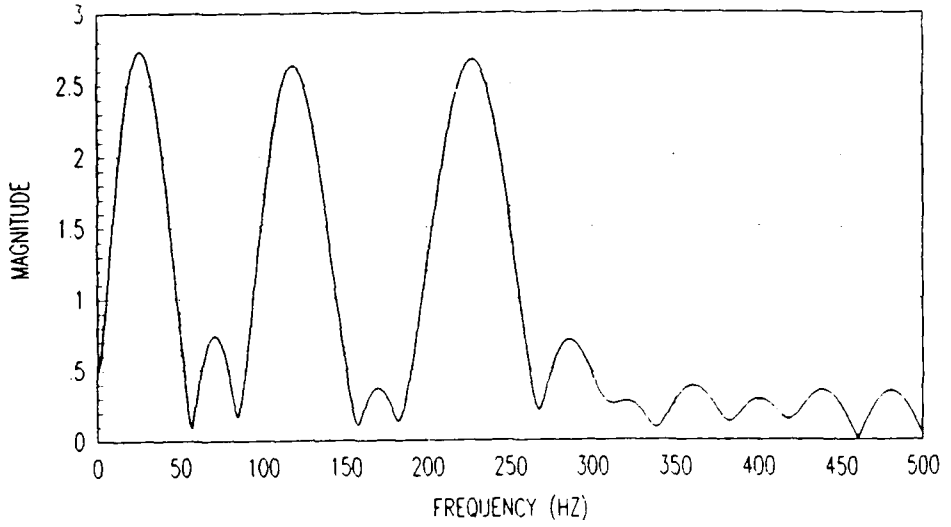

Figure 58. Test 8 - FFT of Output with 30 Hz / 130 Hz / 230 Hz 0.67 Amplitude Input.

Speech Data Noise Reduction

In this final group of tests, the objective was to use real data in an effort to determine if the network could be used for applications outside of a laboratory. Speech data was used to parallel the work done by Tamura and Waibel. The actual words used were the digits 0 through 9, spoken by a thirty-four year old male, and digitized at 8 kHz.

Approach and Assumptions. The networks used in this sequence of experiments had sixty nodes in each layer, providing the same configuration as was used by Tamura and Waibel in their experiments. Data was digitized to 255 discrete values ranging from -128 to 127. Shifting and scaling each data point to create the target output vector was accomplished by adding 128 and then dividing by 255. This ensured that the target output values were always between 0 and 1. However, the maximum amplitude for all of the speech data never exceeded the range -75 to 75, so the actual target output

was always within 0.21 and 0.80. These target values ensured that the network could produce signals whose amplitudes were larger than the target output. In addition, they were approximately the same as the target range used in the single and multiple sine wave testing, so the scaling technique wouldn't cause differences in network performance.

No implementable method was found for computing the signal power of speech data. This prevented using the same signal to noise ratios as in the previous experiments, and also prevented direct comparisons between the experiments conducted here and the ones conducted previously.

Test 9 - Low Sampling Rate, Low Noise. This experiment consisted of adding Gaussian noise with a variance of 5.0, to speech data sampled at 1 kHz. The speech data used was obtained by down sampling the 8 kHz data to 1 kHz. Whenever the end of the data file was reached, the next signal was acquired by moving to the beginning of the file. The variance of 5.0 was selected to ensure the original signal was not corrupted too badly. Since the magnitude for most of the data was around 30, this variance ensured that a high signal to noise ratio was obtained.

Performance measurements taken during training revealed that the network was able to match the original signal fairly well, see Figures 59 and 60. These measurements also indicated the network was still improving its ability to generate the correct output, and that if training had been continued then better performance might have been obtained.

Figures 61 and 62 show test signals, and as can be seen from these graphs the noise corruption was indeed small when compared to the signal amplitude. Further analysis of these graphs revealed that the network was able to filter out a significant amount of the noise. However, the network also damped the amplitude of the original signal during filtering, and when the network was given silence corrupted by noise, the noise reduction was not as apparent and the network appeared to ring, see Figure 63.

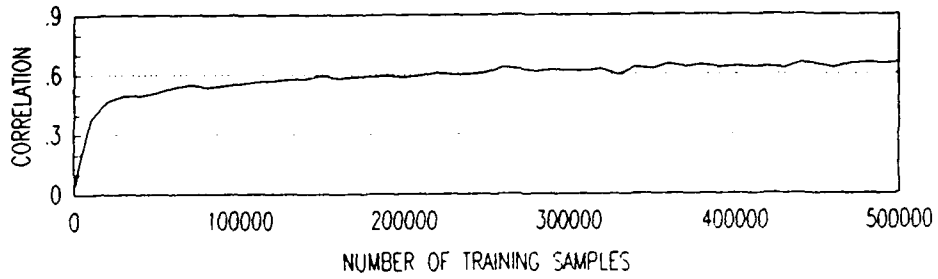

Figure 59. Test 9 - Signal Correlation During Training.

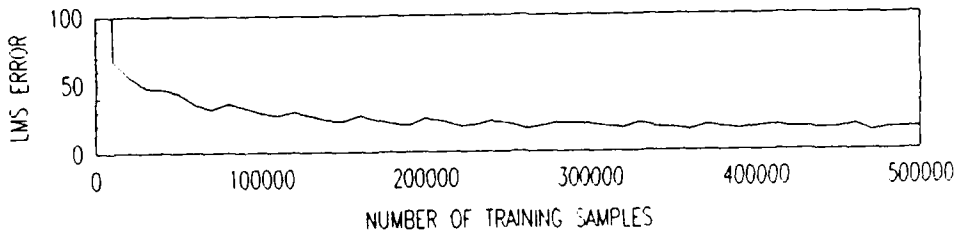

Figure 60. Test 9 - Signal Mean Square Error During Training.

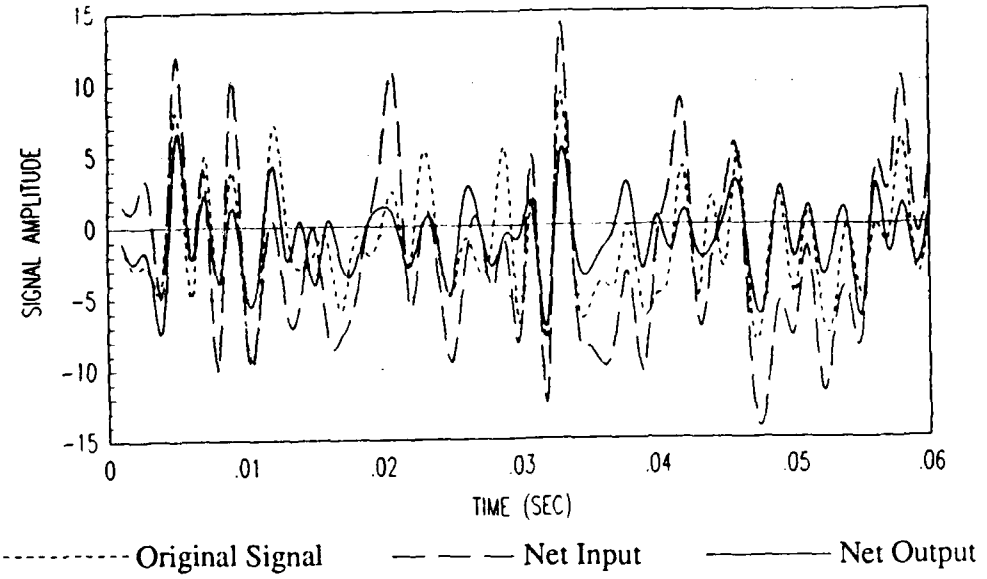

Figure 61. Test 9 - Network Behavior with 1 kHz Speech and Low Noise.

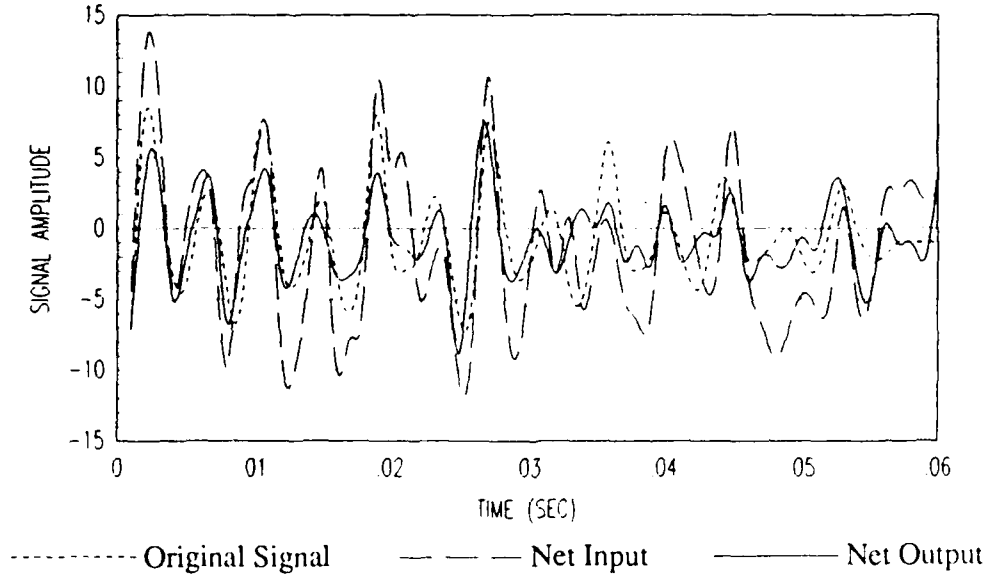

Figure 62. Test 9 - Network Behavior with 1 kHz Speech and Low Noise.

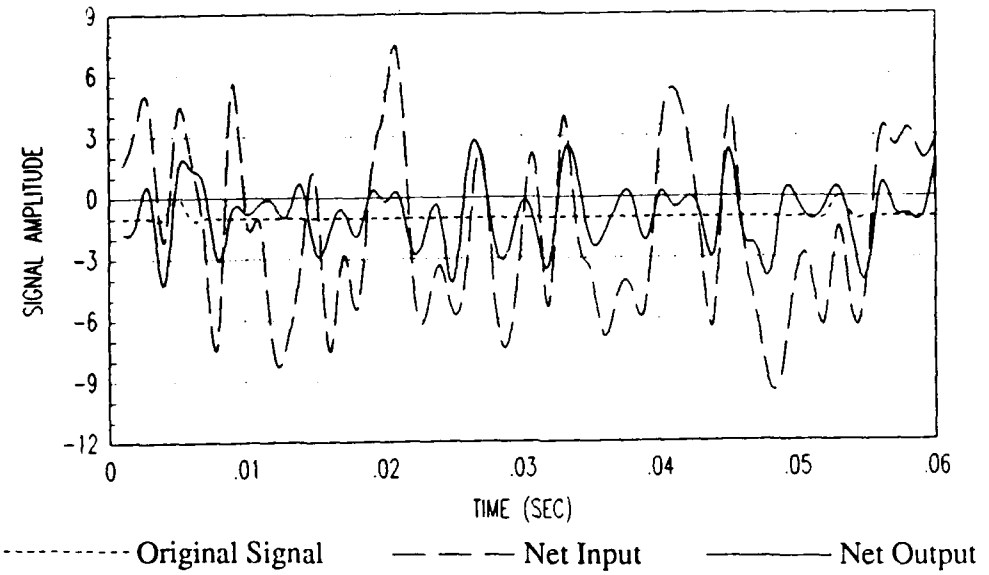

Figure 63. Test 9 - Network Behavior with 1 kHz Speech and Low Noise.

Examination of the test signals' FFT's confirmed that the network was actually preserving the major peaks of the original speech spectrum while filtering noise. The graphs also confirmed that the magnitudes of these frequency components had been reduced; see Figures 64, 65, and 66.

Test 10 - Low Sampling Rate, Moderate Noise. This experiment consisted of adding Gaussian noise with a variance of 50.0, to speech data sampled at 1 kHz. The 1 kHz signals were also obtained by down sampling and, as before, when the end of the data file was reached the next signal was obtained starting at the beginning of the file. Speech signals used in this experiment were to be corrupted by moderate amounts of noise so a variance of 50.0 was selected. Since the largest magnitude for all of the data was around 75, and since the amplitude for most of the data was around 30, this variance ensured that a moderate signal to noise ratio was obtained.

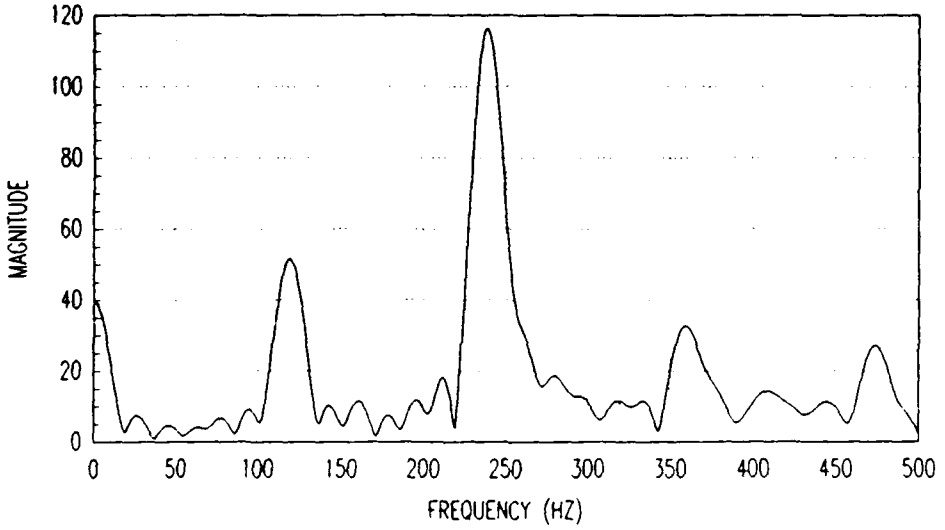

Figure 64. Test 9 - FFT of the Original Signal Shown in Figure 62.

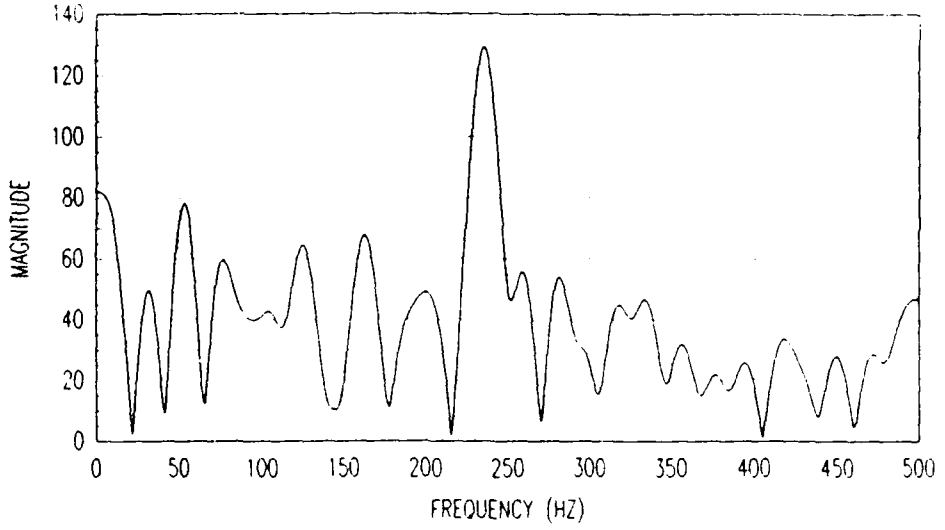

Figure 65. Test 9 - FFT of the Input Signal Shown in Figure 62.

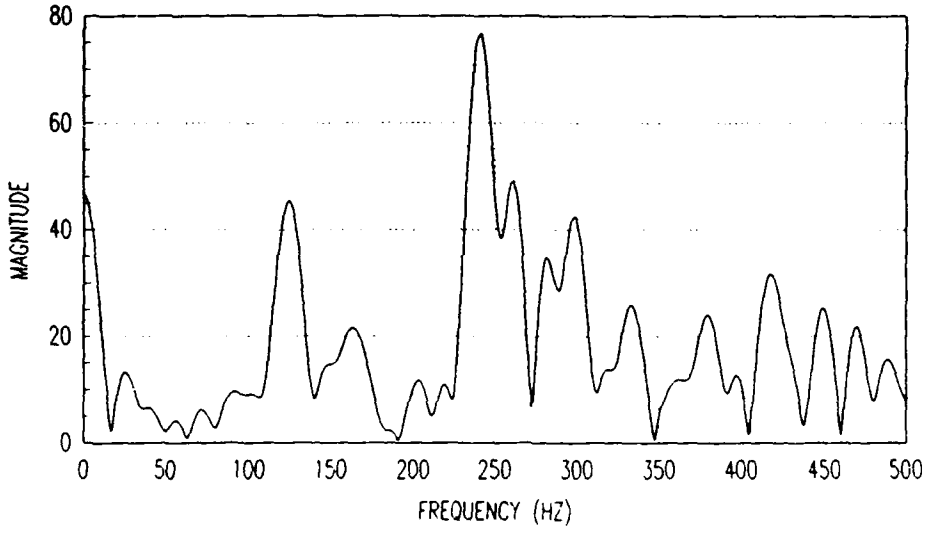

Figure 66. Test 9 - FFT of the Output Signal Shown in Figure 62.

Although the performance measurements showed that the network's accuracy had dropped when compared to Test 9, the drop was small considering there was 10 times the noise, see Figures 67 and 68. In addition, the performance was still getting better when training was terminated.

Figures 69 and 70 provide sample signals from the network. As these graphs show, the network continued to generate a approximation of the original signal. The output followed almost all of the trends of the original signal, but it remained amplitude damped. The signals' FFT's are presented in Figures 71, 72, and 73. All the major frequencies remained intact and at almost exactly the same point as the original signal. The network was able to identify these frequencies even though the noisy input signal contained several other frequencies with a larger magnitude. However, the magnitudes of the main frequency components were much smaller in the output signal than the original signal, and caused an amplitude damped signal to be produced.

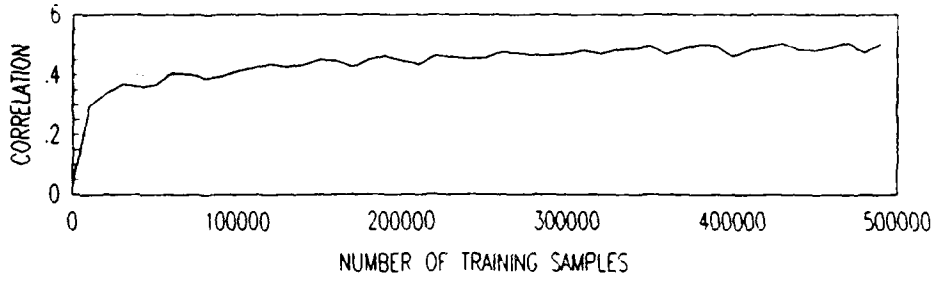

Figure 67. Test 10 - Signal Correlation During Training.

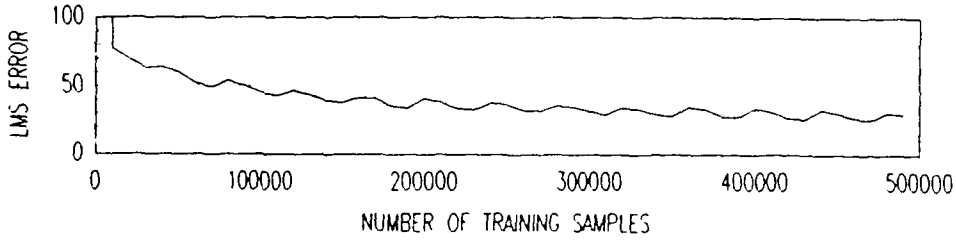

Figure 68. Test 10 - Signal Mean Square Error During Training.

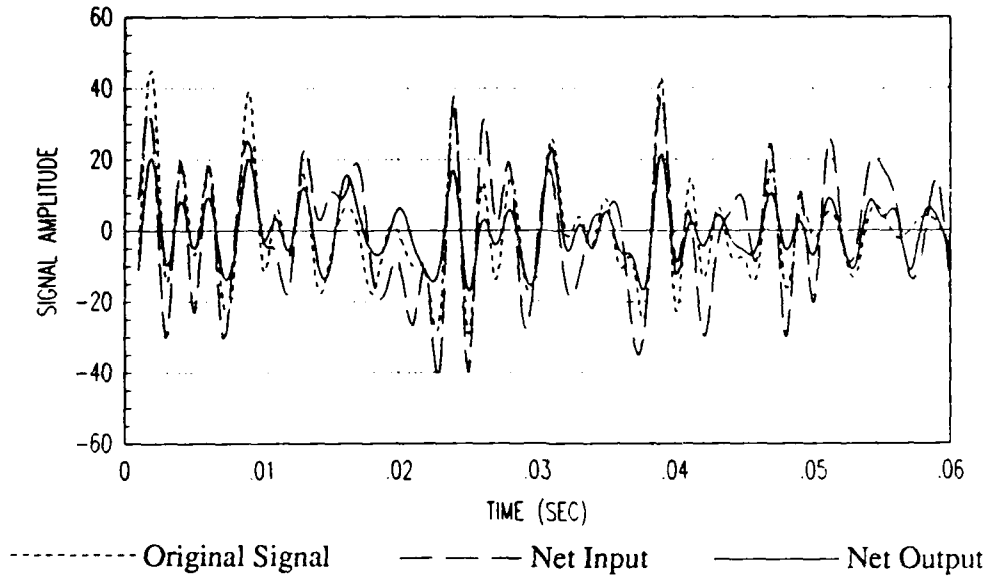

Figure 69. Test 10 - Network Behavior with 1 kHz Speech and Moderate Noise.

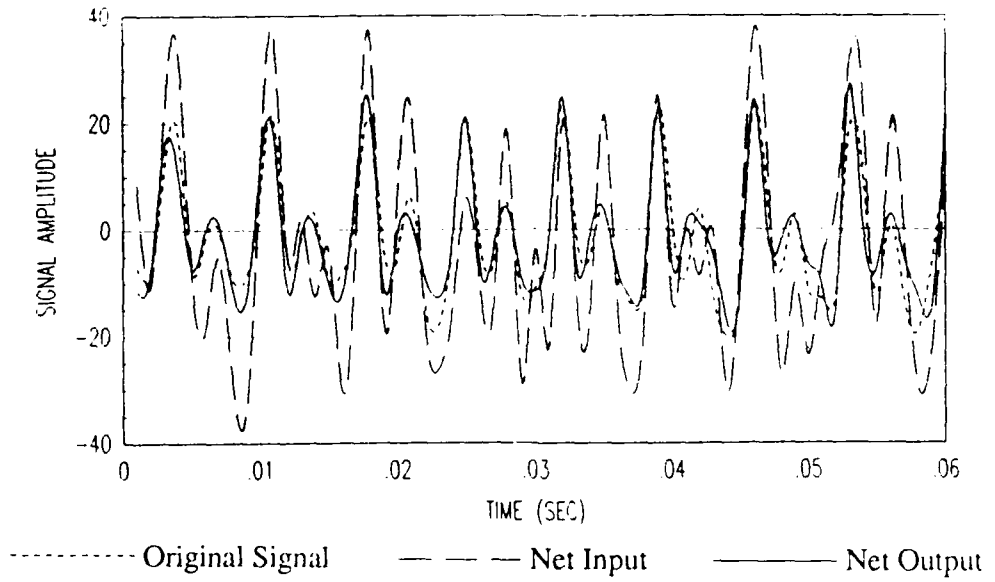

Figure 70. Test 10 - Network Behavior with 1 kHz Speech and Moderate Noise.

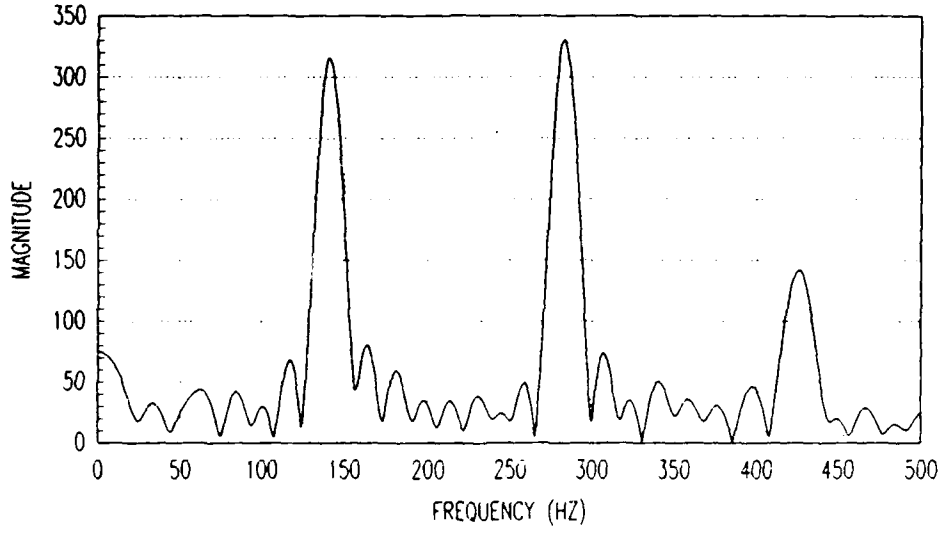

Figure 71. Test 10 - FFT of the Original Signal Shown in Figure 70.

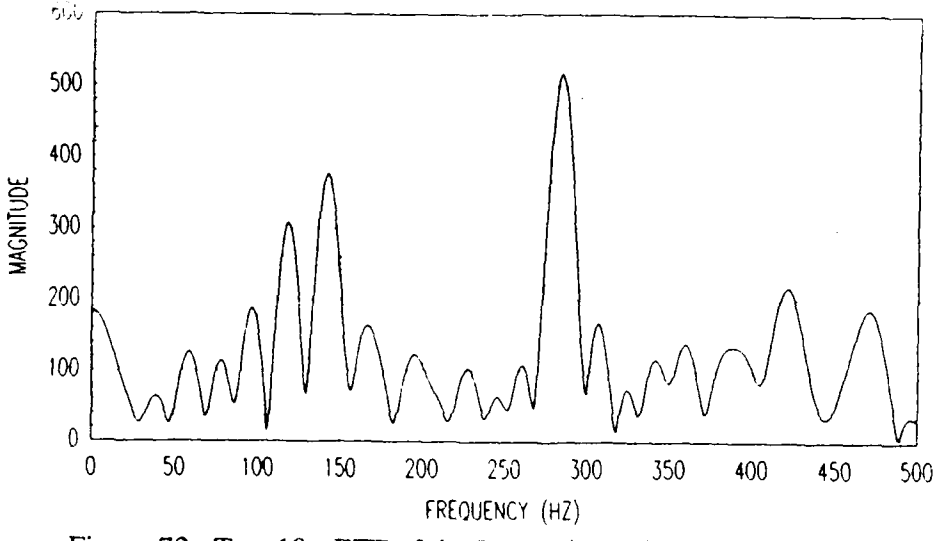

Figure 72. Test 10 - FFT of the Input Signal Shown in Figure 70.

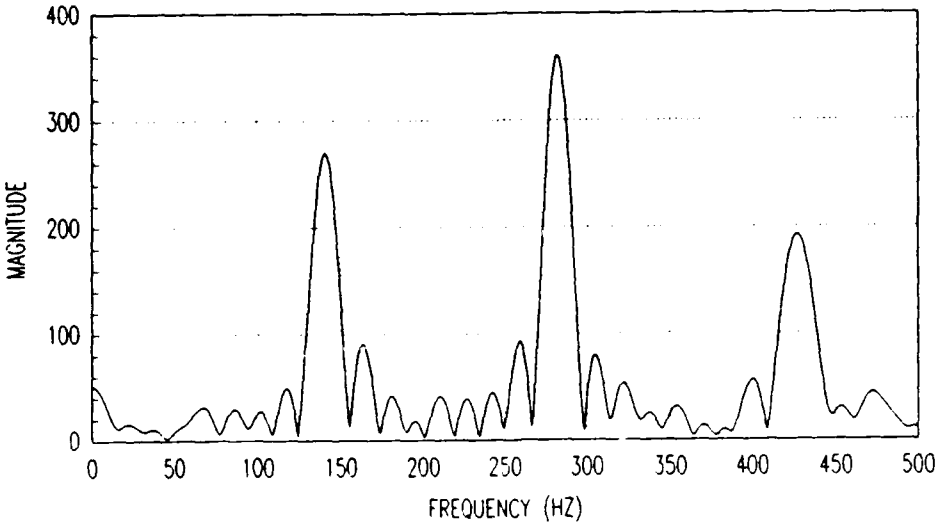

Figure 73. Test 10 - FFT of the Output Signal Shown in Figure 70.

Test 11 - Low Sampling Rate, High Noise. This final experiment consisted of adding Gaussian noise with a variance of 75.0, to speech data sampled at 1 kHz. The speech data used in this experiment had been down sampled from 8 kHz to 1 kHz and each time the end of the data file was reached the next signal was obtained from the beginning of the file. In order to find out how the network behaved with large amounts of noise, a variance of 75.0 was selected to ensure that the signal was badly corrupted. The largest magnitude for all of the speech data was around 75, and since the amplitude for most of this data was around 30, a variance of this magnitude ensured that a small signal to noise ratio was obtained.

Once again there was a drop in the performance measurements taken during training, but considering the noise in this experiment was 50% larger than in Test 10 the performance drop was small. The performance measurements for this network,

however, indicated that the learning curve for the network had probably peaked and that additional training would not have been beneficial, see Figures 74 and 75.

A substantial difference was not seen between the results of this experiment and Test 10. The network was still able to generate an output which approximated the trends of the original signal, see Figures 76 and 77. This was done in spite of the additional noise present in the input signal. Examination of the signals' FFT's revealed that the noisy input had a number of frequencies with large magnitudes, see Figures 78, 79, and 80. This should have made it extremely difficult for the network to identify the fundamental frequencies of the original signal and suppress the remaining ones. Nevertheless, the network was able to locate these frequencies and suppress the noise.

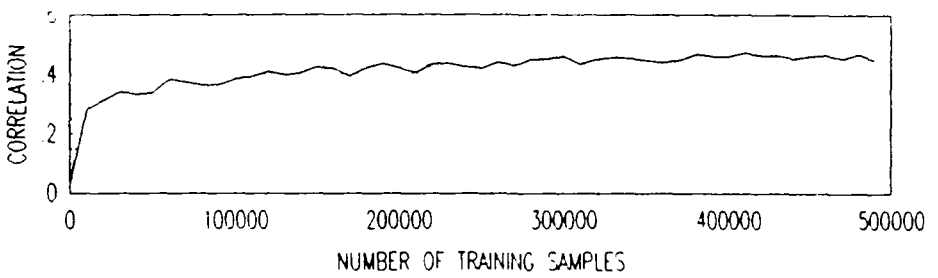

Figure 74. Test 11 - Signal Correlation During Training.

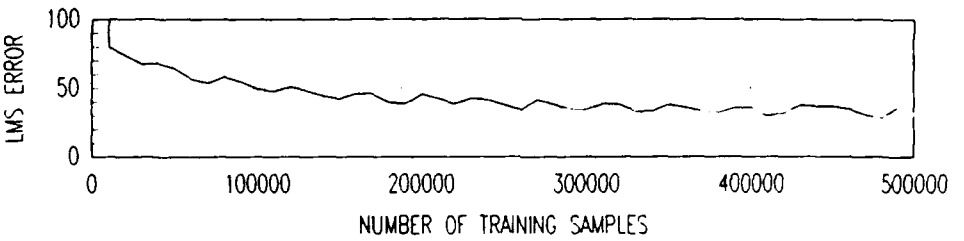

Figure 75. Test 11 - Signal Mean Square Error During Training.

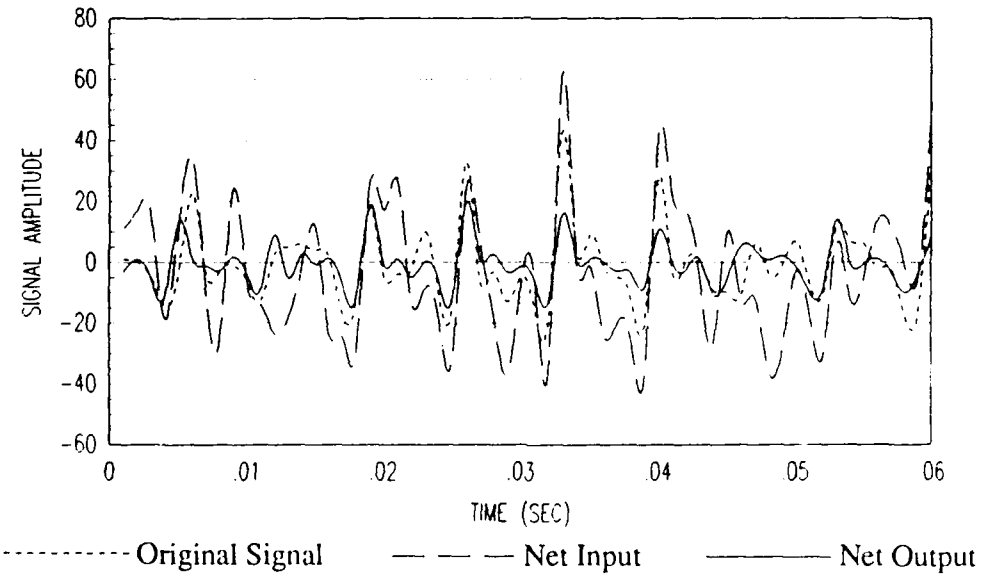

Figure 76. Test 11 - Network Behavior with 1 kHz Speech and High Noise.

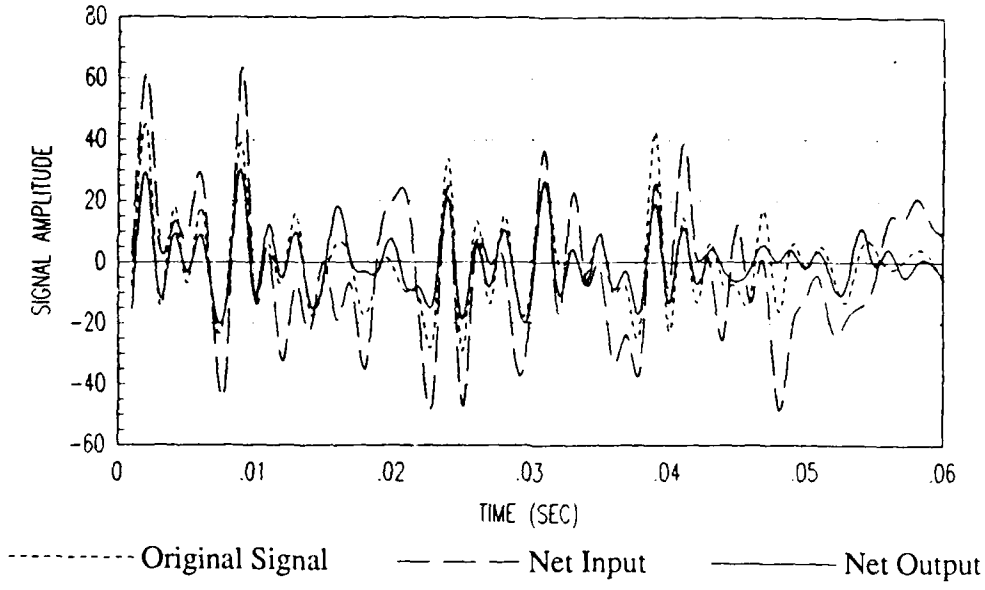

Figure 77. Test 11 - Network Behavior with 1 kHz Speech and High Noise.

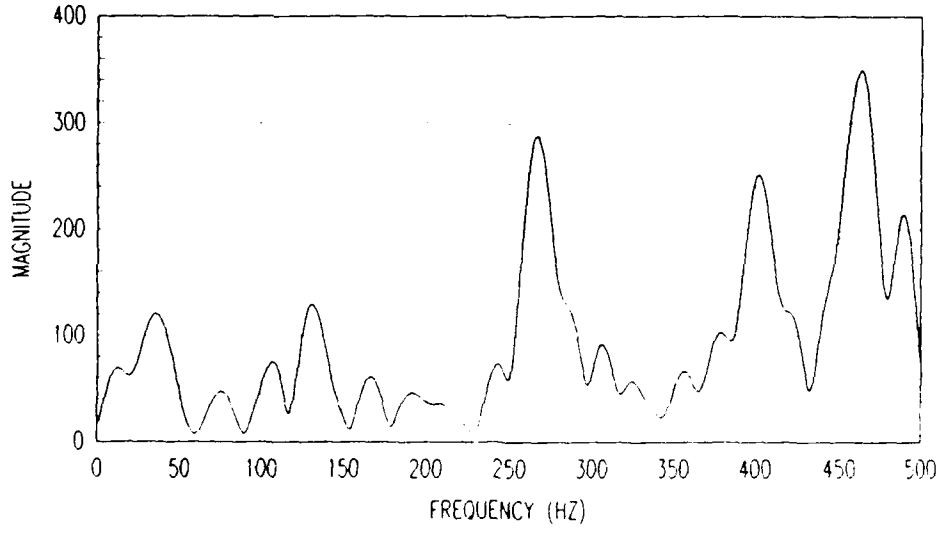

Figure 78. Test 11 - FFT of the Original Signal Shown in Figure 77.

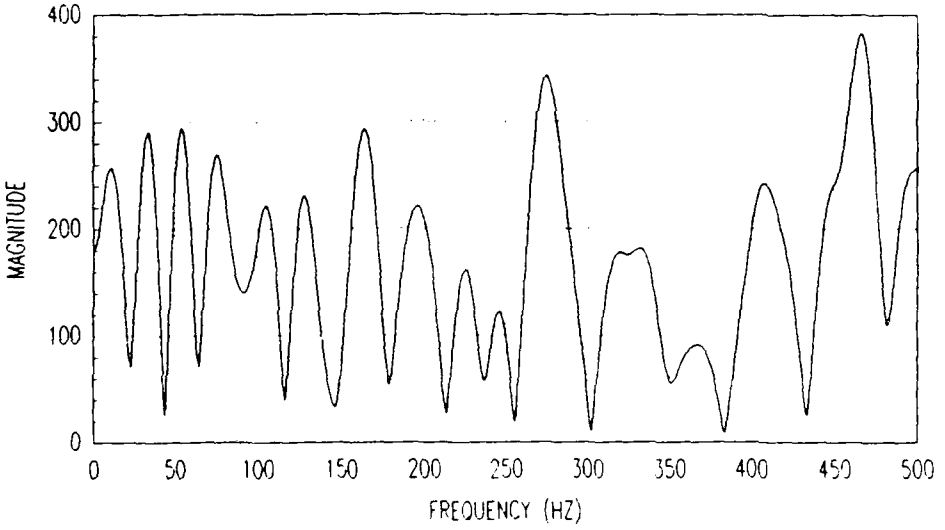

Figure 79. Test 11 - FFT of the Input Signal Shown in Figure 77.

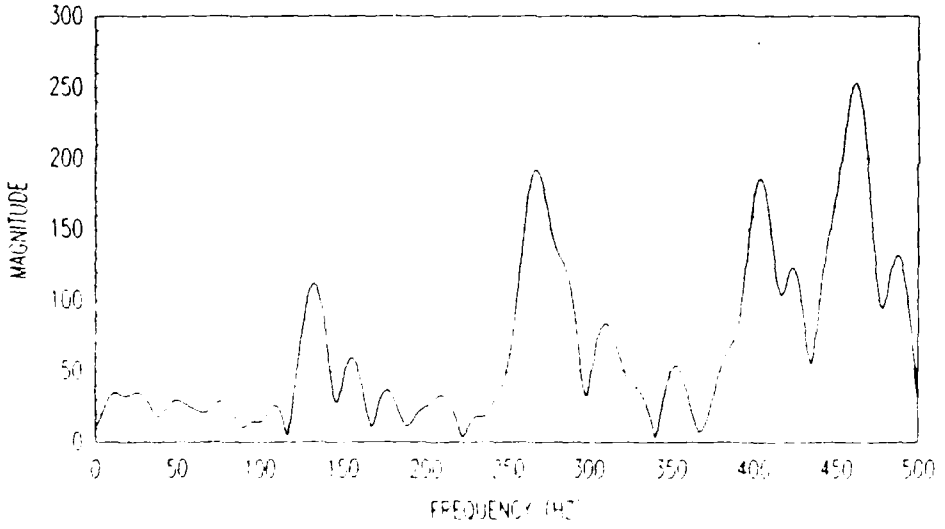

Figure 80. Test 11 - FFT of the Output Signal Shown in Figure 77.

IV. Conclusions & Recommendations

Final Analysis

This research effort was conducted to gain a better understanding of a neural network's ability to filter noise from digitized signals, and to specifically test Tamura and Waibel's assertion that a back-propagation network can reduce the noise component in signals containing human speech. The problem was broken down into three groups of experiments in an effort to identify the filtering properties of a network.

The first group of experiments trained a neural network to filter noise from a single digitized sine wave. Results of these tests showed that the network was indeed able to accomplish the task, usually after only 10,000 training samples. However, the network was only successful when the original signal's frequency was one that had part of the training set. In this case, the network shifted its output frequency to provide an approximation of the original signal. As the original signal's frequency moved outside of the training range, the network's performance diminished and the network began to produce a constant frequency output. This natural frequency was always one of the frequencies used to train the network, and tended to be a frequency around the center of the training range. In addition, as the amount of noise was increased, the network seemed to be more resistant to shifting the output frequency away from the natural one. The network still adjusted the output, but only when the original signal's frequency was further away from the natural frequency. Even then, the network's adjustments were small ones. The phase appeared to be computed by finding the largest input and using that as the point for the peak amplitude of the output signal.

The second group of tests consisted of training a network to filter noise from a signal created by summing three sine waves with different fundamental frequencies. The results from these experiments provided no additional insights, but did support the

previous findings. These networks all developed three natural frequencies, with each one lying approximately in the center of one fundamental training frequency range.

Finally, a neural network was trained to filter noise from actual digitized speech. The results support Tamura and Waibel's assertion that networks can be trained to filter noise from signals containing speech. Comparison of the original speech spectrum with the spectrum of the network's input and output showed that the network preserved major peaks of the speech spectrum while reducing the noise. However, the filtering damped the amplitude of the original signal and the network displayed some ringing when the original signal was silence. One of the most amazing results obtained in the speech tests was that as large increases in noise were applied, the performance of the network did not drop off significantly. This indicated that the network may be able to generate the original waveform from signals with extremely low signal to noise ratios.

Suggestions for Future Research

The results displayed in this research demonstrate that neural networks may be used for applications other than pattern classification. The ability to train neural networks to become filters, suggests that they could be trained to implement other functions as well. For instance, networks may be trained to perform time consuming transform approximations. If successful, then the networks could be etched into silicon and provide very high speed approximations for the transforms. If future research efforts are deemed appropriate, they should be aimed at expanding knowledge concerning how speech filtering is done, as well as, testing the abilities of networks to perform other mathematical functions.

Speech filtering might be improved by shrinking the network size and using a moving window to generate only one output. A odd number of digitized data points, M

though N, would be input to the network. The network's output would be a single value which would correspond to the center input. By moving the window along point by point, the network would be able to generate a output value that corresponded to every input. The possible advantages to this methodology would be that a smaller network might be used to speed up training, and that the resulting output could be more accurate.

Another experiment would be to train a network to compute the Fourier transform for an input pattern. If the network is able to learn to compute the transform, then there would be reason to believe that it might compute other transforms as well.

The number of applications which can be performed by neural networks continues to expand, and now the ability of a network to learn to filter noise from speech signals can be added to the list. Although further research is needed before this type of filter can be used in actual applications, this research has confirmed that a back-propagation neural network can indeed perform the task.

Bibliography

Kabrisky, Mathew. Class Lectures in EENG 621, Pattern Recognition II. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, June-September 1988a.

Kabrisky, Mathew, Professor & Assistant for Advanced Research. Personal Conversations. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, June-September 1988b.

Law, Averill M. and W. David Kelton. Simulation Modeling and Analysis. New York: McGraw-Hill Book Company, 1982.

Lippmann, Richard P. "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, 2: 4-22 (April 1987).

Prescott, Maj Glenn E., Assistant Professor. Personal Conversations. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, July 1988.

Rogers, Steven K. and Mathew Kabrisky. Seminar Lectures, An Introduction to Neural Networks. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, January-June 1988a.

Rogers, Steven K. and James Stright. How Backward Propagation Reduces Error, *Unpublished Paper. School of Engineering, Air Force Institute of Technology*, Wright-Patterson AFB OH, April 1988b.

Ruck, Dennis W. Analysis of Kolmogorov's Theorem. Presentation to EENG 817 students. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, September 1988.

Soukhanov, Anne H., senior editor. Webster's II New Riverside University Dictionary. Boston: Riverside Publishing Company, 1984.

Tamura, Shin'ichi and Alex Waibel. "Noise Reduction Using Connectionist Models," 1988 International Conference on Acoustics, Speech, and Signal Processing. 553-556. New York: IEEE Press, 1988.

Tarr, Captain Greg L. Dynamic Analysis of Artificial Neural Networks Using Simulated and Measured Data. MS Thesis, AFIT/GE/ENG/88D-54. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH.

Widrow, Bernard and Rodney Winter. "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition," Computer, 23: 25-39 (March 1988a).

Widrow, Bernard and others. "Layered Neural Networks for Pattern Recognition", IEEE Transactions on Acoustics, Speech, and Signal Processing, 36: 1109-1118, (July, 1988b)

Ziemer, R. E. and W. H. Tranter. Principles of Communications: Systems, Modulation, and Noise. Boston: Houghton Mifflin Company, 1976.

Vita

Captain Kevin S. Cox [redacted] He graduated from Walter P. Chrysler High School in 1979 and enrolled in Purdue University where he began to study computer science. After being selected to receive a regular commision upon graduation, and to be one of Purdue's Distinguished Military Graduates, he was assigned to the B-1B System Program Office at Wright-Patterson AFB. In 1984 he was joined at Wright-Patterson by [redacted] and they were married in December of the same year. In June 1987 he entered the masters program in the School of Engineering, Air Force Institute of Technology, where he began studying computer engineering.

[redacted]

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCE/ENG/88D-3	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology (AU) Wright-Patterson AFB, OH 45433	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

AN ANALYSIS OF NOISE REDUCTION USING BACK-PROPAGATION NEURAL NETWORKS (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
Kevin S. Cox, Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	88 Dec	80

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Acoustic Filters; Neural Nets; Noise Reduction, Tig... Filters, Speech
06	04		
20	01		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Advisor: Dr. Mathew Kabrisky, Professor of Electrical Engineering
Department of Electrical and Computer Engineering

Abstract on Reverse

10 Jan. 1989

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Dr. Mathew Kabrisky	22b. TELEPHONE (Include Area Code) (513) 255-5276 22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. Abstract

This thesis explored a new approach to filtering noise from digitized signals. A back-propagation neural network was trained to become a filter; and experiments were conducted using single sine wave inputs, multiple sine wave inputs, and human speech inputs. The networks' outputs were then compared to the original signals, and the frequency spectrum was examined to determine the networks' performance.

Results indicated that the networks were indeed able to filter noise. However, the network's filtering ability was strictly limited to signals from the training set. The networks were not able to generalize enough to filter signals whose frequencies had never been encountered. The ability of a back-propagation network to filter noise from actual human speech was particularly interesting, since network performance was not significantly impacted as larger amounts of noise were used to corrupt the input signals.

The conclusion was that back-propagation neural networks can indeed be trained to become digital filters. Keywords: [illegible]